



FAKULTÄT INFORMATIK
INSTITUT FÜR SOFTWARE- UND MULTIMEDIATECHNIK
PROFESSUR MEDIENGESTALTUNG

Diplomarbeit zum Thema

KONZEPTION EINES WEB-FRAMEWORKS ZUR GESTENERKENNUNG AUF BASIS DER BESCHREIBUNGSSPRACHE GEFORMT

zur Erlangung des akademischen Grades

Diplom-Medieninformatikerin

13. Februar 2012

eingereicht von: Dana Henkens
Matrikelnummer: 3249276

Verantwortlicher Hochschullehrer: Prof. Dr.-Ing. habil. Rainer Groh
Wissenschaftlicher Betreuer: Dipl.-Medieninf. Dietrich Kammer

EIDESSTATTLICHE ERKLÄRUNG

Hiermit versichere ich, die vorliegende Arbeit zum Thema

*„KONZEPTION EINES WEB-FRAMEWORKS ZUR GESTENERKENNUNG AUF BASIS DER
BESCHREIBUNGSSPRACHE GEFORMT“*

selbstständig und unter ausschließlicher Verwendung der angegebenen Literatur und
Hilfsmittel erstellt zu haben.

Dresden, 13. Februar 2012

Dana Henkens

DANKSAGUNG

Ich möchte mich an dieser Stelle bei Prof. Dr. Groh und Dietrich Kammer für die Ermöglichung dieser Arbeit bedanken. Vielen Dank für die engagierte und kompetente Betreuung. Herzlichen Dank auch an alle Korrekturleser für die hilfreichen Anmerkungen.

Ein besonderer Dank gilt meiner Familie, die mich während meines gesamten Studiums uneingeschränkt unterstützt hat und immer an mich geglaubt hat. Meinen Freunden und Kommilitonen danke ich für ihren Zuspruch und Ansporn. Ein großer Dank geht an Martin, der sich stets um mein persönliches Wohl während der Arbeitszeit gesorgt hat.

INHALTSVERZEICHNIS

1	Einleitung	3
1.1	Motivation	3
1.2	Zielstellung	3
1.3	Gliederung	4
2	Grundlagen	5
2.1	Web Interface	5
2.1.1	Entwicklung und Wandel	5
2.1.2	Web-Browser	6
2.1.3	Kompatibilität und Standardkonformität	7
2.2	Gestenbasierte Interaktion	7
2.2.1	Klassifikation von Eingabegeräten	8
2.2.2	Eigenschaften und Struktur zweidimensionaler Gesten	9
2.2.3	Klassifikationsmerkmale	10
2.3	Softwaretechnologische Grundlagen	12
2.3.1	Clientseitige Webentwicklung	12
2.3.2	Inhalte und Struktur mit (X)HTML	12
2.3.3	Gestaltung und Adaption mit CSS	13
2.3.4	Programmiersprache JavaScript	14
2.3.5	JavaScript Coding-Muster	15
2.3.6	Ereignisverarbeitung	18
3	Verwandte Arbeiten	21
3.1	Architekturmodell	21
3.2	Web-Browser	23
3.2.1	Verarbeitung von Multitouch- und Stift-Eingaben	23
3.2.2	TUIO-basierte Eingabeverarbeitung	25
3.3	Interpretation von Gesten	27
3.3.1	Web-Frameworks	27
3.3.2	Sprachbasierte Frameworks	30
3.3.3	Methoden der Gestenerkennung	36
3.4	Applikation	42
3.4.1	Vergleich von Maus-, Stift- und Touch-Interaktion	42
3.4.2	Besonderheiten des Multitouch-Interface	44
3.4.3	Aktionen und Anweisungen	44

3.4.4	Anwendung von Gesten	46
3.4.5	Bewertung des derzeitigen Stands touchbasierter Web-Anwendungen	50
4	Synthese	51
4.1	Entwicklung und Anwendung einer mobilen Web-Anwendung	51
4.2	Webkonforme Gestenformalisierung	53
4.2.1	Ereignisprofil atomarer Gesten	53
4.2.2	Formalisierung von Mausgesten	54
4.2.3	Kontakttypen	54
4.2.4	Sequentielle Komposition	55
4.2.5	Geltungsbereich und Objektkontext	57
4.2.6	Aktionsbeschreibung und -zuordnung	60
4.2.7	Konfigurationsparameter	61
4.2.8	Potenzial der Wiederverwendung von Definitionen	62
4.2.9	Anomalien	63
4.2.10	Zusammenfassung	64
5	Konzeption und Implementierung	67
5.1	Anforderungskriterien	67
5.2	Komponenten des Frameworks	68
5.2.1	Globale Konstanten	69
5.2.2	Parser	69
5.2.3	Selector Engine	70
5.2.4	Ereignisverarbeitung	71
5.2.5	Datenmodell	72
5.2.6	Template-Erzeugung und Gestenerkennung	73
5.2.7	Feedback	76
5.3	Anwendung	78
5.3.1	Installation	78
5.3.2	Anwendungsschnittstellen	78
5.3.3	Testumgebung	80
5.3.4	Anwendungsprototyp	82
6	Zusammenfassung und Fazit	85
6.1	Ausblick	86
	Literaturverzeichnis	i
	Abbildungsverzeichnis	ix
	Tabellenverzeichnis	xi
	Listingverzeichnis	xii

1 EINLEITUNG

1.1 MOTIVATION

Die Gestaltung von Webseiten ist maßgeblich durch Eingabegeräte wie die Maus und Tastatur geprägt. Mit dem Trend zur Mobilität wachsen die Interaktionstechniken des Internets über ihre klassische Domäne hinaus. Web-Anwendungen werden immer mehr über berührungssensitive Benutzungsschnittstellen, wie sie in Smartphones und Tablets verbreitet sind, bedient. Im Zuge dieser Entwicklung werden Programmierschnittstellen standardisiert, die es Webentwicklern ermöglichen auf Berührungen des Touch-Displays zu reagieren und darauf aufbauend Multitouch-Gesten in Webseiten zu implementieren. Das daraus entstehende Potenzial für web-basierte Software ist noch weitgehend unerforscht und erfordert praktische Anwendungsbeispiele. Eine Hürde bei der Entwicklung innovativer Bedienkonzepte stellt jedoch die verhältnismäßig komplizierte Umsetzung einer Gestenerkennung dar. Es existieren Bibliotheken, die sich des Problems annehmen. Deren Fokus liegt jedoch in der Bereitstellung vordefinierter anwendungstauglicher Gesten oder sind meist auf einen bestimmten Anwendungsbereich spezialisiert. Es fehlt demnach an generischen Ansätzen, die es Webentwicklern erlauben anwendungsspezifische Gesten ebenso einfach zu implementieren, wie die Struktur und das Layout einer Webseite. Grundlage eines solchen Ansatzes ist eine Gestenbeschreibungssprache, die sich aus verarbeitbaren Regeln zusammensetzt und darauf aufbauend erlaubt, den Erkennungsprozess von der Konzeption einer Geste zu trennen.

1.2 ZIELSTELLUNG

Aufbauend auf der Gestenbeschreibungssprache GeForMT (Gesture Formalization for Multi-Touch) [Kammer et al., 2010] ist es das Ziel dieser Diplomarbeit ein Gestenerkennungs-Framework für Web-Anwendungen zu konzipieren. In Hinblick auf eine plattformübergreifende Verwendung wird dabei auf standardisierten Web-Technologien aufgebaut. Dies beinhaltet auch die Betrachtung einer geräteübergreifenden Eingabeverarbeitungsmethodik, die eine umfassende Unterstützung der in Web-Anwendungen verwendeten Eingabegeräte und Interaktionstechniken gewährleisten soll. Hierzu ist zu analysieren, inwiefern sich die Maus-, Touch- und Stiftinteraktion aufeinander abbilden lassen und welche softwaretechnischen Ansätze in Hinblick auf eine Generalisierung existieren. Des Weiteren ist zu untersuchen, wie die Interoperabilität zwischen der formalen Gestenbeschreibung und bestehenden Standards der Webprogrammierung herzustellen ist. Dies erfordert ein tieferes Verständnis der Konventionen der verfügbaren Programmierschnittstellen. Neben der soft-

waretechnischen Realisierung ist vor allem auch die praxistaugliche Anwendung von Gesten auf Elementen der Web-Benutzeroberfläche von Interesse und wird aufbauend auf der Recherche beispielhafter Implementierungen diskutiert. Den Schwerpunkt dieser Arbeit bildet die Entwicklung eines Verfahrens für die Erkennung der formal definierten Gesten basierend auf primitiven Eingabedaten. Dazu sind bestehende Erkennungsalgorithmen für den Einsatz in einer webbasierten Umgebung zu bewerten und auf ihr Potenzial hinsichtlich Erweiterungen zu prüfen, die den Anforderungen des Formalisierungsansatzes in GeForMT genügen. Aufbauend auf den genannten theoretischen Überlegungen ist die Implementierung des Framework-Konzepts vorzunehmen und dessen Funktionsweise anhand von Beispielen praktisch zu testen.

1.3 GLIEDERUNG

Die vorliegende Arbeit gliedert sich in sechs Hauptkapitel, deren Inhalte in diesem Abschnitt für einen Überblick kurz zusammengefasst werden. Nachdem das Arbeitsthema in diesem ersten Kapitel eingeleitet wurde, werden nachfolgend in Kapitel 2 die Grundlagen der Webentwicklung und der gestenbasierten Interaktion behandelt. Neben der Klärung wiederverwendeter Begriffe werden hier vor allem softwaretechnische Grundlagen erklärt, die im weiteren Verlauf der Arbeit vorausgesetzt werden und für die Konzeption des Frameworks von Bedeutung sind. Der Aufbau von Kapitel 3 orientiert sich an den wesentlichen Verarbeitungsschritten von Gesten. Das zugrunde liegenden Architekturmodell webbasierter Software wird in Kapitel 3.1 erläutert. Darauf aufbauend werden Ansätze der Eingabedatenverarbeitung (Kapitel 3.2) und anschließend verwandte Arbeiten aus dem Bereich der Formalisierung und der methodischen Erkennung von Gesten vorgestellt (Kapitel 3.3). Das Kapitel schließt mit der Auseinandersetzung von Aspekten des Interaktionsdesigns auf Anwendungsebene ab (Kapitel 3.4). In der Synthese (Kapitel 4) werden die gewonnenen Erkenntnisse aufgegriffen und ein Anwendungsszenario formuliert, in dem der praktische Nutzen eines Gestenerkennungs-Frameworks in Web-Anwendungen dargelegt wird. Darüber hinaus werden in Kapitel 4.2 Ansätze der Gestenbeschreibungssprache GeForMT im Kontext der Webentwicklung diskutiert. In diesem Rahmen werden Optimierungs- und Erweiterungsvorschläge entwickelt. Kapitel 5 beschäftigt sich anschließend mit der konkreten softwaretechnologischen Umsetzung des Gestenerkennungs-Frameworks. An dieser Stelle werden Entwurfsentscheidungen einzelner Module und deren Beziehungen zueinander erläutert, sowie die Installation und Anwendung des Frameworks erklärt. Der Implementierungsteil schließt mit einem Anwendungsprototypen ab. Das letzte Kapitel (Kapitel 6) fasst die Ergebnisse zusammen und gibt einen Ausblick für die Weiterentwicklung der Arbeit.

2 GRUNDLAGEN

2.1 WEB INTERFACE

Der Begriff *Web Interface* wird in dieser Arbeit für die Benutzerschnittstelle (engl.: *User Interface*) einer Software verwendet, die über einen *Web-Browser* (näheres dazu in Kapitel 2.1.2) zugänglich ist. Das *Web* ist demnach ein Dienst des Internets, der *Web-Seiten* (verwendet wird auch der Begriff *Web-Anwendung*) auf Basis des *Hyper-Text Transfer Protocol* (HTTP) ausliefert.

2.1.1 Entwicklung und Wandel

1989 schlägt Tim Berners-Lee am CERN¹ den *Global Hypertext Space* ([Lessiak, 2008]) vor. Ursprüngliche Idee des Projektes war, wissenschaftliche Dokumente über ein Netzwerk austauschbar zu machen und zu verknüpfen. Das erste Hypertext-Programm wird im darauffolgenden Jahr unter dem Namen *WorldWideWeb* entwickelt. [Dumke et al., 2003, S. 14] Viele Unternehmen erkannten das Potenzial eines globalen Netzwerks und so folgten Browser-Implementierungen wie *Mosaic*² (1993), *Netscape Navigator* (1994) und der *Internet Explorer* von Microsoft (1995).

*Anfänge und
Entwicklung des Webs*

Mit Erscheinen des Internet Explorers entbrannte ein Wettstreit um die führende Marktposition hinsichtlich der Browser-Entwicklung. Eine Folge der Auseinandersetzung zwischen Netscape und Microsoft ist die Vernachlässigung von Web-Standards. [Lessiak, 2008] Stattdessen implementierten beide Seiten proprietäre Erweiterungen um die Konkurrenz abzuordnen. Sowohl die Spezifikationen der Web-Technologien und Bibliotheken entwickelte sich auseinander (vgl. Kapitel 2.3.4), als auch die Auslegung dieser für die Web-Seitenvisualisierung. Der Konflikt endete 1998 mit dem Ausscheiden von Netscape. [Dumke et al., 2003]

Browserkriege

Aus dieser historischen Entwicklung heraus erhob sich eine Initiative (siehe Abschnitt 2.1.3), die die browserübergreifende Funktion des Webdienstes durch die Standardisierung der Web-Technologien ermöglichen sollte. Der Wettstreit der Browser-Hersteller erfolgt seitdem zunehmend um die Einhaltung standardisierter Spezifika-

Standardisierung

¹CERN (Conseil Européen pour la Recherche Nucléaire - z.d.t. Europäische Organisation für Kernforschung) nennt sich eine Forschungseinrichtung in Genf in der Schweiz

²Mosaic wurde 1993 im National Center for Supercomputing Applications an der Universität Illinois als erster Browser mit grafischer Benutzeroberfläche für verschiedene Betriebssysteme entwickelt. [Lessiak, 2008, S. 3]

tionen. Auf dieser Basis entstehen Bibliotheken und Frameworks, die die Entwicklung von Web-Anwendungen auf einem höheren Abstraktionsniveau erlauben. [Lesiak, 2008, S. 6]

Wandel der Nutzung und Wahrnehmung

Der Fortschritt zeigt sich besonders im Nutzungswandel. Der Computer wird zum alltäglichen universellen Werkzeug, wodurch sich verschiedenste Interessen bezüglich der Verwendung der zentralen Plattform herausbilden. Das anfänglich reine Informationsmedium entwickelte sich zu einem interaktiven Raum, in dem Nutzer kollaborativ Inhalte konsumieren und produzieren. Diese neue Generation wird auch unter dem Schlagwort *Web 2.0* zusammengefasst. Darüber hinaus entwickeln sich Techniken, die die Information des Mediums semantisch in Beziehung setzen, was zu einer Vernetzung auf höherer Ebene führt. Daraus folgt unter anderem eine zunehmende Personalisierung und Adaptierung gegenüber dem Nutzungskontext. Es entstehen immer komplexere Web-Applikationen, die Dienste bereitstellen und mit denen sich Geschäftsprozesse zentral abwickeln lassen. (vgl. [Cooper et al., 2007, S. 174 ff.])

Allgegenwärtigkeit

Mit der fortschreitenden Entwicklung der Mobilfunktechnik gewinnt das mobile Internet an Bedeutung. Es ermöglicht einen standortungebundenen Zugang zu Webinhalten. Man spricht auch von *Ubiquität* (Allgegenwärtigkeit). [Bieh, 2008, S. 159] Die Nutzung des Webs über mobile Endgeräte erfordert jedoch neue Adaptions-techniken und Bedienkonzepte, die der eingeschränkten Leistungsfähigkeit, wie der geringeren Rechenkapazität und Bildschirmauflösung gerecht werden. (vgl. [Bieh, 2008, S. 27 f.])

Berührungssensitive Eingabetechniken

Insbesondere die fehlende Zeigersteuerung in kleinen Mobilgeräten erschwert die Interaktion mit Web-Seiten, die ursprünglich für eine Bedienung mit der Maus ausgelegt sind. Apple gelang 2007 die erfolgreiche Markteinführung des iPhones, das allein durch Berührung des Displays zu steuern ist, und setzte damit den Maßstab für kommende Mobilgeräte. (vgl. [Bieh, 2008, S. 27 f.])

2.1.2 Web-Browser

Desktopbasierte und mobile Web-Browser

Der Web-Browser bildet einen grafischen als auch funktionalen Rahmen um das Web-Interface und bietet grundlegende Funktionen an, um in Web-Seiten zu navigieren. Kern der Anwendung ist die *Rendering-Engine*, die für die grafische Umsetzung der Web-Seiten-Dokumente verantwortlich ist. Namhafte Web-Browser sind *Firefox*, *Internet Explorer*, *Safari*, *Chrome* und *Opera* (vgl. [Wikipedia, 2012]). Für den mobilen Einsatz in Smartphones und Tablets haben die Hersteller kompaktere Versionen ihrer Browser entwickelt, die speziell an die Erfordernisse der Geräte angepasst sind. Sie basieren auf Betriebssystemen wie zum Beispiel *iOS* [Apple, 2012] oder *Android* [Google, 2011].

Grundfunktionen

Zur funktionalen Grundausstattung der Web-Browser zählen im Allgemeinen die Adresszeile, die Einbindung eines Web-Suchdienstes, Funktionen zum Vor- und Zurückblättern in der sogenannten *Browser-History*, sowie eine Ablage für favorisierte oder zuletzt besuchte Seiten. Verbreitet ist auch das Feature, das grafische Interface des Browsers zu verstecken. (Vollbildfunktion) Das Web-Interface erscheint dadurch als alleinstehende Anwendung in einem Raum, der allein durch den Display-Rahmen begrenzt ist.

2.1.3 Kompatibilität und Standardkonformität

Das *World Wide Web Consortium* (W3C)³ ist eine Organisation für die internationale Standardisierung von Web-Technologien. Ihre Ziele formuliert sie mit "Web for All" und "Web on Everything". Zum einen setzt sie sich für die Internationalisierung und Bedienbarkeit von Web-Seiten verschiedener Anwendergruppen ein. Zum anderen tritt sie dafür ein, das Web auf verschiedensten Plattformen und Systemen einheitlich zugänglich zu machen. Jede Organisation kann eine Mitgliedschaft im W3C beantragen und aktiv auf die Entwicklung der Empfehlungen Einfluss nehmen.

World Wide Web Consortium

Aufgrund schleppender Entwicklungen der Standards und Uneinigkeit über die Arbeit des W3C im Zeitraum um 2004 gründeten die führenden Browser-Hersteller die Arbeitsgruppe *WHATWG*⁴ (Web Hypertext Application Technology Working Group), die um die Akzeptanz der Browser-Hersteller und einen schnelleren Fortschritt bemüht ist. Sie begannen einen unabhängigen Entwurf eines HTML-Standards zu erarbeiten, der dem W3C unterbreitet wurde. (siehe Kapitel 2.3.2) Die beiden Initiativen arbeiten dennoch eng zusammen. [Schäfer, 2011b]

Initiative der Browser-Hersteller

Die Kompatibilität von Web-Anwendungen ist damit maßgeblich davon abhängig, wie gut die Web-Browser die Empfehlungen des W3C umsetzen. Um dies festzustellen ist eine Prüfung gegen die Spezifikationen notwendig. 1998 entwickelte Todd Fahrner den sogenannten *Acid-Test*, an dem sich Browser-Hersteller messen können. Grundlage des Tests ist eine Referenzgrafik, die es möglichst fehlerfrei zu rendern gilt. Auf dieser Basis folgten weitere Versionen, die vom *WaSP*⁵ (Web Standards Project) veröffentlicht wurden und weitere Richtlinien des W3C aufnahmen. [Wikipedia, 2011] Aktuelle Tests ergeben eine breite Unterstützung der Standards in den zuvor genannten Browsern. (vgl. [Wikipedia, 2011])

Acid-Test

2.2 GESTENBASIERTE INTERAKTION

Interaktion ist nach [Lessiak, 2008, S. 14] ein zielorientierter Dialog, der aus einer wechselseitigen Abfolge von Aktion und Reaktion der Handlungspartner besteht. Im Fokus dieser Arbeit steht der Dialog zwischen Mensch und System, in dem die Geste das Mittel der Verständigung ist.

Begriff: Interaktion

Die *Geste* ist eine Form der non-verbalen Kommunikation und als bedeutungstragende Handlung im Dialog zu verstehen. (vgl. [Lessiak, 2008]) In dieser Arbeit wird der Begriff auf zweidimensionale Gesten eingegrenzt. Betrachtet werden damit Handlungen, die auf einer definierten Fläche erfolgen.

Begriff: Geste

Die wesentlichen Eigenschaften zweidimensionaler Gesten werden in Abschnitt 2.2.2 genauer betrachtet. Zuvor werden in Abschnitt 2.2.1 die für dieses Thema relevanten Eingabegeräte vorgestellt. In Abschnitt 2.2.3 wird anschließend eine Klassifikation anhand bedeutungsunterscheidender Merkmale von Gesten vorgenommen.

Gliederung der folgenden Abschnitte

³World Wide Web Consortium - URL: <http://www.w3.org/> [Stand: 29.09.2011]

⁴Web Hypertext Application Technology Working Group - URL: <http://www.whatwg.org/> [Stand: 26.01.2012]

⁵Web Standards Project - URL: <http://www.webstandards.org/> [Stand: 26.01.2012]

2.2.1 Klassifikation von Eingabegeräten

Dieser Abschnitt gibt einen Überblick über die Eingabegeräte, die üblicherweise im Umgang mit einem Web-Interface eingesetzt werden. Im Fokus liegt dabei das Potenzial der Gesteninteraktion.



Abbildung 2.1: Beispiele von Eingabegeräten für die Interaktion mit Webinhalten

Maus Die klassische Maus ist in Desktop-Computersystemen die am meisten verbreitete Eingabeperipherie. Durch sie können relative Bewegungen von Hand und Arm übertragen werden. Sie verfügt üblicherweise über zwei Fingertasten, die mit den Zuständen für Links- und Rechtsklick belegt sind. Zum Scrollen von Ansichten hat sich das Maus-Rad durchgesetzt.

Trackpad Das Trackpad ist ein Eingabefeld, das häufig in Notebooks unterhalb der Tastatur integriert ist und zu Mobilitätszwecken die klassische Maus ersetzt. Ähnlich wie bei der Maus-Steuerung werden relative Bewegungen in Zeigerbewegungen übersetzt. Zusätzlich werden Tasten angeboten, um das Interaktionspotenzial der Maus nachzubilden. Moderne Trackpads erlauben darüber hinaus die Eingabe von Multitouch-Gesten, wie zum Beispiel das Scrollen mit zwei Fingern. Das *Magic Trackpad* [Apple, 2010] ist ein Beispiel eines externen Gerätes dieser Art. (siehe Abbildung 2.1a)

Multitouch-Maus Neue Entwicklungen wie die Multitouch-Maus (beispielsweise die *Magic Mouse* [Apple, 2009b]) kombinieren die Eingabetechniken der klassischen Maus und des Trackpads in einem Gerät. Die Oberfläche der Maus reagiert auf Berührungseingaben und ermöglicht Touch-Gesten, die hauptsächlich für das Blättern und Scrollen von Ansichten reserviert sind.

Touchscreen-Monitor Zu den Touchscreens zählen im Allgemeinen Displays, deren visueller Ausgabebereich eine unauffällige Schicht von Sensoren integriert, die direkte Eingaben auf dem Bildschirm registrieren. Das Potenzial der Gesten-Eingabe hängt stark von der verwendeten Hardware und Messtechnik ab. Ein Überblick über die verschiedenen Multitouch-Technologien wird in [Teiche et al., 2009] und [ELO TouchSystems, 2011] gegeben. Unter der Bezeichnung Touchscreen-Monitor werden in dieser Arbeit berührungsempfindliche Displays zusammengefasst, die in Desktopsystemen eingesetzt werden oder auch in Notebooks integriert sind, um die Touch-Bedienung optional neben der Steuerung über die Standardeingabegeräten zu erlauben. (vgl. Abbildung 2.1b)

Tablets- und Smartphones Insbesondere in mobilen Endgeräten wie Smartphones (vgl. Abb. 2.1d), PDAs (Personal Digital Assistant), Tablets (vgl. Abb. 2.1c) und ähnliche Handheld⁶-Geräte hat sich

⁶Handheld: Engl. Bezeichnung für ein tragbares Mobilgerät, das in einer Hand gehalten werden kann

die Multitouch-Technologie mit steigender Rechenkapazität verbreitet. Die Hardware-Tastatur wird allmählich von den flexibleren softwaregestützten Eingabekonzepten, die sich mit Touchscreens offenbaren, ersetzt. [Weil, 2010] Der Zugang zum Internet über eine entsprechende Browsersoftware gehört mittlerweile zu den Standardfunktionen, mit denen die Geräte ausgeliefert werden. Zusätzlich zu den Touchscreen-Sensoren, verfügen moderne Geräte über Bewegungssensoren, die die Ausrichtung erfassen. Verbreitet ist die Unterscheidung zwischen Hochkant- und Querformat, wonach die Darstellung optimiert wird.

Eine weitere erwähnenswerte Eingabetechnik ist die stiftbasierte Eingabe. Stifte dienen als Werkzeug für eine präzisere Eingabe, das vor allem in Grafikbearbeitungs- und Schreibsoftware eingesetzt wird. Nicht selten werden in den im vorigen Abschnitt beschriebenen mobilen Endgeräten zusätzlich Stifte angeboten. (vgl. Abb. 2.1d) Vorallem bei sehr kleinen Displays kommen sie als Eingabehilfe zum Einsatz. Spezialisierte Grafik-Tablets verfügen über Möglichkeiten der sensorischen Erkennung, die über das reine registrieren eines Kontaktes hinausgehen. Basierend auf Induktion und elektromagnetischer Resonanz können Druck, Stiftneigung, Ausrichtung und axiale Drehung erfasst werden. (vgl. [Dorau, 2011])

Stiftbasierte Systeme

2.2.2 Eigenschaften und Struktur zweidimensionaler Gesten

Die *kommunikative Absicht* einer Geste, die in der Einführung dieses Kapitels beschrieben wurde, formuliert DORAU als eine von vier elementaren Eigenschaften. Gesten sind darüber hinaus an einen bestimmten *Geltungsbereich* gebunden, haben ein *Rückmeldeverhalten* und lassen sich durch ihr charakteristisches *Ereignisprofil* und anhand distinktiver Merkmale spezifizieren. [Dorau, 2011, S.170 ff.]

Haupteigenschaften

Das Ereignisprofil wird in [Dorau, 2011, S. 172] als Choreografie einer Geste beschrieben, die aus einer Kette elementarer Ereignisse besteht und in ihrer Gesamtheit ein charakteristisches Muster der Ausführung bilden. Nach [Wu et al., 2006] und [Wigdor und Wixon, 2011] lässt sich eine Geste in zeitlich abgrenzbare Phasen einteilen: Die Registrierung, Ausführung und der Abschluss einer gestischen Handlung.

Profil einer Geste

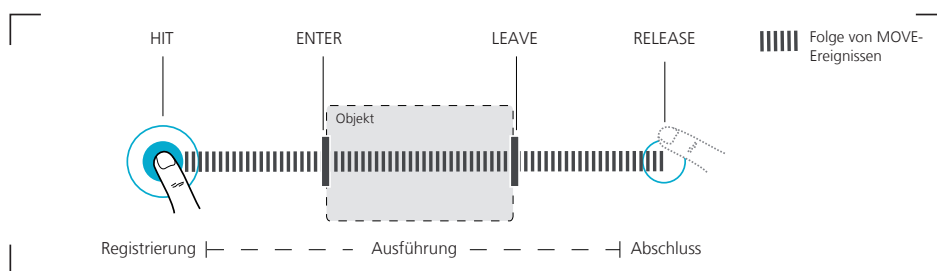


Abbildung 2.2: Einheiten des Ereignisprofils einer zweidimensionalen Geste

Die Einheiten eines Ereignisprofils sind wie in Abbildung 2.2 dargestellt einzuordnen und werden im Folgenden erläutert. (vgl. [Dorau, 2011, S.174 ff.]) Der Begriff *Kontakt* wird dabei als verallgemeinerte, geräteunspezifische Bezeichnung für eine registrierte Aktion verwendet.

Elementare Ereignisse

- **HIT (Berühren):** Es wird ein Kontakt hergestellt, der je nach Eingabemodalität der Berührung auf einer sensitiven Oberfläche, dem Drücken einer Maustaste oder einer bestimmte Hand-Konfiguration (bei berührungslosen Eingabesystemen) entspricht.

- *ENTER (Eintreten)*: Das Eintreten bezieht sich auf einen räumlichen Geltungsbereich, an dessen Rändern das Ereignis ausgelöst wird, sobald eine Bewegung in den Bereich hineinführt.
- *MOVE (Bewegen)*: Ausgelöst durch die Veränderung der Position, repräsentiert das Ereignis eine Bewegungseinheit. Eine Sequenz von Bewegungs-Ereignissen beschreibt einen *Bewegungspfad*.
- *LEAVE (Austrreten)*: Das Ereignis ist vergleichbar mit ENTER, setzt aber voraus, dass ein Kontakt innerhalb des räumlichen Geltungsbereiches besteht und eine Bewegung nach außen stattfindet.
- *RELEASE (Loslassen)*: Der Kontakt geht verloren, indem der Finger vom Touchscreen oder der Maustaste abgehoben wird, oder sonstige Systementscheidungen zum Abbruch der Geste führen.

Einfachste Geste

Aus der Sequenz der elementaren Ereignisse formt sich das charakteristische Ereignisprofil einer Geste. Die einfachste Geste besteht aus der Ereignis-Sequenz HIT-RELEASE. Im Sinne einer Touch-Eingabe ist dieses Ereignisprofil einer TAP-Geste zuzuordnen. Als Maus-Geste würde die Bezeichnung CLICK zutreffen.

2.2.3 Klassifikationsmerkmale

Objektorientierte und freie Gesten

Jede Geste bezieht sich entweder auf ein Objekt, das in Folge einer Aktion manipuliert wird oder hat zumindest einen räumlichen Geltungsbereich, in dem sie erkannt wird. Nach [Dorau, 2011, S. 170] werden demnach *objektorientierte* und *freie Gesten* unterschieden. Im Allgemeinen umfasst der Geltungsbereich für freie Gesten die gesamte Benutzeroberfläche und bildet eine virtuelle Eingabeschicht, die über allen Inhalten und Bedienelementen zu liegen scheint. In [Wobbrock et al., 2009] wird eine ähnliche Klassifikation vorgenommen, die darüberhinaus Mischformen unterscheidet. Dies sind Gesten, deren Umgebungsparameter in Beziehung zu setzen sind (*world-dependent*) oder sowohl objektorientiert als auch frei sind (*mixed dependencies*), wie es beispielsweise bei einer zweihändigen Geste der Fall sein kann.

Einzelpfad-Gesten und zusammengesetzte Gesten

Es kann verallgemeinert werden, dass eine Geste mit einem HIT-Ereignis beginnt und einem RELEASE-Ereignis endet. Das bedeutet allerdings nicht, dass ein RELEASE-Ereignis bereits Anlass ist, die Geste auszuwerten und zu interpretieren. Das Ereignisprofil einer DOUBLE-TAP-Geste setzt sich beispielsweise aus der Ereignissequenz HIT-RELEASE-HIT-RELEASE zusammen. Um das Ereignisprofil einer Geste korrekt abbilden zu können, müssen also neben räumlichen Relationen auch zeitliche Parameter berücksichtigt werden. Für die DOUBLE-TAP-Geste bedeutet das, dass nach dem ersten RELEASE-Ereignis in einem gewissen Zeitrahmen ein weiteres Ereignis erwartet wird. Dieser Zeitrahmen wird auch *Kontiguitätsintervall* genannt. [Dorau, 2011, S. 196] Damit findet sich ein weiteres Merkmal einer zweidimensionalen Geste: Eine *Einzelpfad-Geste* ist im Rahmen der Ereignisse HIT und RELEASE definiert. Eine Geste kann sich unter Berücksichtigung des Kontiguitätsintervalls aus einer sequentiellen Abfolge von Einzelpfad-Gesten zusammensetzen (*Zusammengesetzte Geste*).

Statische und dynamische Gesten

Am Beispiel der DOUBLE-TAP-Geste zeigt sich der Fall, dass eine Geste nicht zwangsläufig durch eine Bewegung charakterisiert sein muss. Zu unterscheiden sind nach [Saffer, 2008] *statische* und *dynamische Gesten*, gleichbedeutend mit der Klassifizierung von *punkthaften* und *pfadbasierten Gesten* nach [Henzen, 2010]. Pfadbasierte Gesten zeichnen sich durch ihre *Bewegungsform* aus, repräsentiert durch einen Bewegungspfad. Eine Bewegungsform ist demnach ein charakteristisches räumliches

Muster eines Bewegungspfades. Nach [Dorau, 2011, S. 178 f.] können folgende Bewegungsformen unterschieden werden:

Bewegungsformen

- *Translatorische Bewegung*: Eine eindimensionale, geradlinige Bewegung.
- *Wellenförmige Bewegung*: Eine Bewegung, die einem frequenten Muster folgt.
- *Kreisförmige Bewegung*: Eine Bewegung, die gleichmäßig um einen zentralen Punkt herumgeführt wird.
- *Spiralförmige Bewegung*: Eine kreisförmige Bewegung mit dynamischen Abstand zum Kreiszentrum.
- *Spur*: Nach [Dorau, 2011] wird eine Spur als "Abfahren" eines Bewegungspfades definiert. Sie ist als komplexe Form aufzufassen, die sich aus verschiedenen Bewegungsformen zusammensetzen kann.

Räumliche Muster sind nicht nur innerhalb eines Bewegungspfades elementarer pfadbasierter Gesten festzustellen. Durch die Kombination elementarer Gesten bilden sich Relationen zwischen den jeweiligen Eigenschaften und Bewegungsmerkmalen. Diese lassen sich als weitere Charakteristik der Geste erschließen. Eine Geste die sich beispielsweise aus mehreren TAP-Gesten zusammensetzt und bewusst unterschiedliche Positionen fokussiert, folgt in ihrer Gesamtheit einem räumlichen Bewegungsmuster. Das selbe gilt für die Komposition pfadbasierter Gesten.

*Räumliche Muster
kombinierter Gesten*

Abhängig vom Erkennungsspektrum des Eingabesystems können ein oder mehrere Kontakte gleichzeitig detektiert werden. Bedeutend ist die Anzahl der Kontakte. In [Weil, 2010] werden darauf aufbauend *1- bis 5-Level-Gesten* betrachtet, die systemabhängig mehr oder weniger Relevanz haben. Während 1-Level-Gesten die Grundlage für Zeigehandlungen bilden und einen hohen Kompatibilitätsgrad verschiedener Eingabegeräte aufweisen, erweitern Level-2-Gesten das Gestenrepertoire nicht nur quantitativ, sondern auch qualitativ um neue Steuerungsmöglichkeiten, wie die Objektmanipulation oder ähnliche kontextbezogene Eingaben. [Dorau, 2011, S. 174] Höherwertige Gesten sind nach [Weil, 2010] in größeren Systemen vertreten und erst bei einem ausreichend großen Touchscreen sinnvoll. *Multitouch-Gesten* können ein- oder beidhändig ausgeführt werden. Dies hat sich jedoch bisher nicht als unterscheidbares Merkmal durchgesetzt, sondern vertritt Anweisungen meist in größeren Dimensionen. [Wobbrock et al., 2009] Bezüglich der Anzahl registrierter Kontakte ergeben sich viele Variationen einer Geste (vgl. [Hinrichs und Carpendale, 2011]), dessen prinzipielle Ausführung und Intention oftmals die selbe ist.

*Single- und
Multitouch-Gesten*

In [Wobbrock et al., 2009] wird eine semantische Unterscheidung von Gesten bezüglich ihrer "Natur" vorgenommen. *Symbolische Gesten*, wie zum Beispiel ein Pfeil oder ein Fragezeichen, sind bildlich zu interpretieren. *Physikalische Gesten* bilden das Verhalten bei der Manipulation von realen Objekten ab. In [Kammer et al., 2010] wird darüber hinaus unterschieden, ob eine physische Geste eine Zeigehandlung darstellt (*deiktisch*), objektorientiert ist (*manipulierend*) oder als freie Geste auf die gesamte Ansicht Einfluss nimmt (*navigierend*). Das Auseinanderbewegen mehrerer Finger auf einem Objekt führt beispielsweise zu dessen Skalierung, während die selbe Geste applikationsweit ein Zoomen in die dargestellte Szene bewirkt. Als *metaphorisch* wird eine Geste bezeichnet, die ein Verhalten abstrahiert und imitiert. Dabei ist zu beachten, dass Metaphern je nach kulturellem Hintergrund verschieden ausgelegt werden können. [Kammer et al., 2010] Eine *abstrakte Geste* erklärt sich hingegen nicht über eine Metapher oder ein Symbol und muss folglich erst erlernt werden. Ein Beispiel nach [Wobbrock et al., 2009] ist das dreimalige Tippen auf ein Objekt, um es zu löschen.

*Semantische
Klassifikation*

Online- und Offline-Gesten

Ein entscheidender Aspekt für das Rückmeldeverhalten von Gesten ist der Zeitpunkt ihrer Interpretation. WOBBROCK ET AL. unterscheiden eine *diskrete* und *kontinuierliche* Erkennung (vgl. [Wobbrock et al., 2009]), gleichbedeutend mit der Klassifikation von *Offline*- und *Online*-Gesten in [Kammer et al., 2010]. Letztere fordern ein stetiges Feedback während der Ausführung und werden daher unmittelbar und fortlaufend interpretiert. Offline-Gesten hingegen werden einmalig erkannt nachdem die Ausführung abgeschlossen ist.

2.3 SOFTWARETECHNOLOGISCHE GRUNDLAGEN

2.3.1 Clientseitige Webentwicklung

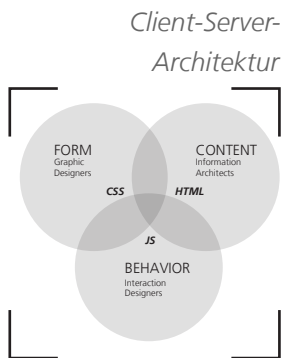


Abbildung 2.3: Design-Disziplinen nach [Cooper et al., 2007, S. xxxi] und zugeordnete Web-Standards

Web-Anwendungen sind in Regel auf Basis einer *Client-Server-Architektur* implementiert. Ein Webserver im Internet stellt die Anwendung bereit und der Client (Web-Browser) nimmt Daten auf Anfrage hin in Empfang. Die vorliegende Arbeit konzentriert sich auf die clientseitige Programmierung. Das bedeutet, dass die Ausführung der Anwendungslogik auf dem Endgerät erfolgt. [Dumke et al., 2003] Diese Entscheidung liegt darin begründet, dass die Verarbeitung von Nutzereingaben eine auf Performanz ausgerichtete Architektur erfordert. Die zeitintensive Kommunikation zwischen Client und Server beschränkt sich damit auf die vom Nutzer bewusst initiierten Anfragen, wie der Wechsel einer Seite.

Die Konzeption einer Anwendung erfolgt nach [Cooper et al., 2007] in drei Disziplinen (vgl. Abb. 2.3), die zusammen ein optimales *Bedienerlebnis*⁷ zum Ziel haben. Dies geht mit der Auffassung nach [Dumke et al., 2003] einher, dass die technische Trennung von Inhalt, Gestaltung und Logik zum guten Programmierstil gehört. Entsprechend kommen Web-Standards wie (X)HTML (siehe Abschnitt 2.3.2), CSS (siehe Abschnitt 2.3.3) und JavaScript (siehe Abschnitt 2.3.4) zum Einsatz, die den jeweiligen Entwicklungsbereichen zuordenbar sind.

2.3.2 Inhalte und Struktur mit (X)HTML

Historische Entwicklung

Die *Hypertext Markup Language* (HTML) ist eine Auszeichnungssprache, um die Inhalte einer Web-Seite zu strukturieren und semantisch einzuordnen. [Dumke et al., 2003] Der erste Entwurf auf Basis der Metasprache SGML (Standard Generalized Markup Language) geht auf die Entwicklung des Hypertext-Systems von Tim Berners-Lee zurück. (vgl. Kapitel 2.1.1) Die Version HTML 4.01 [W3C, 1999], die 1999 vom W3C verabschiedet wurde, ist nach Aussage in [Robbins, 2010] die am weitesten verbreitete Spezifikation. Es folgte eine Neuformulierung der Auszeichnungssprache auf Basis der strengeren Regeln von XML (Extensible Markup Language), genannt *XHTML* (Extensible Hypertext Markup Language). In dem gemeinsamen Projekt des W3C und der WHATWG wird aktuell an HTML 5 als Nachfolger-Spezifikation zur HTML-Version 4 gearbeitet, die voraussichtlich 2014 verabschiedet wird. (vgl. [W3C, 2011e])

⁷User Experience (z.d.t. Bedienerlebnis) ist ein von Donald A. Norman geprägter Begriff als Designaspekt interaktiver Anwendungen. [Dorau, 2011]

Die Auszeichnungen erzeugen Elemente der Benutzeroberfläche, die sich wie folgt grob klassifizieren lassen: *Strukturelemente* zeichnen Seitenbereiche und inhaltliche Zusammenschlüsse aus. Sie werden demnach vor allem für die Gruppierung von Elementen verwendet. Eine spezielle *Datenstrukturierung* kann mit Listen und Tabellen vorgenommen werden. Während Text ein elementares Medium in Webdokumenten ist, das auch ohne Auszeichnung eingebettet werden kann, stellt die Sprache *Medien-Elemente* bereit, mit denen sich hauptsächlich Bilder, Audio- und Videoinhalte integrieren lassen. Aber auch Sub-Anwendungen wie zum Beispiel Applets lassen sich durch Konstrukte dieser Elemente-Klasse einbetten, sofern der Web-Browser das Format unterstützt. Zu den *interaktiven Elementen* zählen vor allem Link-Auszeichnungen und seit HTML 5 unter anderem auch Zeichenflächen und dynamische Menüs. Eine weitere wichtige Gruppe in Bezug auf die Nutzereingabe stellen die *Formularelemente* dar. Eine umfassende und tiefer greifende Beschreibung aller möglichen HTML-Elementen wird in [Robbins, 2010] gegeben.

Zusammenfassung von HTML-Elementen

2.3.3 Gestaltung und Adaption mit CSS

Die Gestaltung der in HTML definierten Inhalte wird in Form von *Cascading Style Sheets* (CSS) vorgenommen. Der Begriff *Cascading* steht für das Prinzip der Vererbung und Überlagerung der Stil-Eigenschaften. Listing 2.1 zeigt den allgemeinen Aufbau einer CSS-Regel.

CSS-Regel

```
1 Selector {
2   property: value;
3   ..
4 }
```

Listing 2.1: Allgemeiner Aufbau einer CSS-Regel

Die Schnittstelle zu HTML bildet dabei der *Selector* - ein formales Suchmuster, anhand dessen ein oder mehrere HTML-Elemente ausgewählt werden, um ihnen Stil-Eigenschaften zuzuweisen. Die verarbeitende Softwarekomponente wird auch *Selector-Engine* genannt. Tabelle 2.1 fasst die wichtigsten Selektor-Typen zusammen.

Selektoren

Bezeichnung	Beschreibung	Beispiel
Typ-Selektor	Selektiert Elemente anhand des Tag-Namen	body, div
Klassen-Selektor	Selektiert Elemente mit dem class-Attribut 'myClass'	.myClass
Id-Selektor	Selektiert Elemente mit der Id-Attributwert 'myId'	#myId
Universal-Selektor	Gilt für jedes Element	*

Tabelle 2.1: Wichtige CSS-Selektoren im Überblick

In einer Regel können mehrere Selektoren, durch Komma getrennt, aufgezählt werden. Darüber hinaus gibt es Attributselektoren (Einschränkung der Anfrage anhand

Selektor-Typen

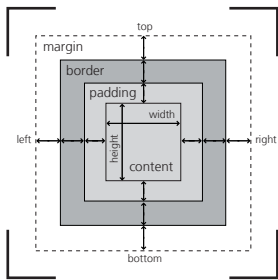


Abbildung 2.4:
Box-Modell eines
Web-Elements

Darstellungsoptimierung
mit Media-Queries

von Attributwerten), Kombinatoren (Selektion anhand von Verwandtschaftsbeziehungen in der Element-Hierarchie) oder Pseudoklassen wie zum Beispiel der Ausdruck `:link`, um unbesuchte Verweise herauszufiltern.

Ein wichtiges Konzept für die Formgestaltung von Elementen ist das *Box-Modell* (vgl. Abb. 2.4). Der Inhalt (*content*) wird von den Boxen *padding*, *border* und *margin* umgeben, mit denen jeweils der Abstand zum Rahmen, die Rahmengröße und der Abstand zu den umgebenden und übergeordneten Elementen in den vier möglichen Seiten genau bestimmt werden kann.

Ein weiteres hilfreiches Konzept für die Adaption des Layouts an verschiedene Ausgabemedien ist das sogenannte *Media-Query*. Ein Media-Query ist ein logischer Ausdruck, der die Bedingungen des Ausgabemediums festlegt, für die die enthaltenen CSS-Regeln bestimmt sind. Das Layout kann an die Voraussetzungen des Gerätes angepasst werden, indem Größe, Auflösung, Farbsystemeigenschaften, Format und Medientyp abgeprüft werden. (vgl. [W3C, 2010a]) Das Beispiel in Listing 2.2 zeigt, wie das Layout aufbauend auf einem Media-Query für ein mobiles Endgerät im Querformat optimiert werden kann.

```
1 @media handheld and (max-device-width: 480px)
2     and (orientation: landscape) { .. }
```

Listing 2.2: Beispiel eines Media-Queries für die optimierte Darstellung auf
einem mobilen Endgerät

2.3.4 Programmiersprache JavaScript

ECMAScript

JavaScript ist eine Programmiersprache, die 1995 von Netscape eingeführt wurde. Netscape bemühte sich um die Verbreitung der Sprache als Standard und spannte dazu die Organisation ECMA (European Computer Manufacturers Association) mit ein. Der Sprachkern von JavaScript wurde in der Spezifikation ECMA-262 zum Standard deklariert und ist unter dem Namen *ECMAScript* bekannt. [Schäfer und Strübig, 2007] Die aktuelle Ausgabe von ECMAScript ist in der Version 5 veröffentlicht.⁸

Document Object
Model

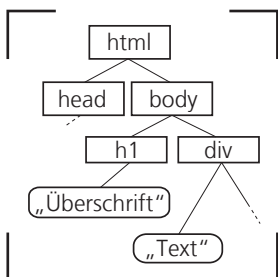


Abbildung 2.5:
Repräsentativer
DOM-Baum eines
HTML-Dokuments

Darüber hinaus ist eine Standardisierung des Zugriffs auf die Elemente des Web-Interface notwendig. Die Browserhersteller entwickelten zunächst eigene Modelle (sogenanntes *dynamisches HTML - DHTML*), die jedoch weit auseinander gingen und nicht kompatibel waren. 1998 schaltete sich das W3C mit einer offiziellen Empfehlung für ein standardisiertes Modell mit der Bezeichnung *Document Object Model* (DOM) ein. Die JavaScript-Version, die DOM erstmals umsetzt, ist die Version 1.5. [Schäfer und Strübig, 2007]

DOM definiert eine Programmierschnittstelle für den Zugriff auf Elemente von HTML- und XML-Dokumenten und das zugrundeliegenden Objektmodell. Das Objektmodell bildet die Elemente des Interface durch hierarchisch in Beziehung gesetzte Knoten ab. (vgl. Abbildung 2.5) Im Mittelpunkt stehen dabei die Knoten vom Typ **Element** (in der Grafik durch Rechtecke markiert). Für den Knotenbaum des Dokumentes wird im Folgenden die Bezeichnung *DOM-Baum* verwendet. [Dumke et al., 2003, S. 88]

⁸Aktuelle ECMAScript-Spezifikation unter der URL <http://www.ecma-international.org/publications/standards/Ecma-262.htm> [Aufruf: 14.Sept.2011]

Unter JavaScript sind demnach verschiedenste Programmierschnittstellen (APIs) zusammengefasst, die im Kern auf ECMAScript basieren. Als Skriptsprache kommt JavaScript ohne eine Vorübersetzung, wie es beispielsweise bei JAVA üblich ist, aus. Skripte werden in Form von Quelltext ausgeliefert und unmittelbar interpretiert. [Dumke et al., 2003, S. 235] Sie werden der Reihenfolge ihrer Einbindung entsprechend ausgeführt. Konsequenz dessen ist, dass ein Skript auf Objekte vorher eingebundener Skripte Zugriff haben kann, jedoch nicht auf nachfolgende. [Schäfer, 2011a] Weitere Eigenschaften der Sprache sind der objektorientierte klassenlose Ansatz und das Vererbungskonzept über Prototypen (siehe Abschnitt 2.3.5). Bis auf fünf primitive Datentypen (**number**, **string**, **boolean**, **null**, **undefined**) werden alle Datenstrukturen, selbst Funktionen, als Objekte beschrieben, die wiederum Eigenschaften und Methoden besitzen. Zu unterscheiden sind *native Objekte* und *Host-Objekte*. Native Objekte sind entweder von ECMAScript bereitgestellte Objekte (**Object**, **Array**, **Function** u.w.) oder selbst definierte Objekte. Host-Objekte hingegen werden von der Browser-Umgebung bereitgestellt (DOM-Objekte, **window**, **document** u.w.). [Stefanov, 2011]

JavaScript-
Eigenschaften

2.3.5 JavaScript Coding-Muster

STEFANOV erklärt, dass wichtige Entwurfsmuster wie *Strategy*, *Factory* oder *Observer* in einer schwach typisierten Sprache wie JavaScript nur schwer abzubilden sind. Interessanter erscheint daher ein Einblick in die sogenannten *Coding-Muster* zu geben, die JavaScript-spezifisch sind und sich für bestimmte Probleme bewährt haben. [Stefanov, 2011, S. 2]

Prinzipiell werden eigene Objekte durch **new Object()** erzeugt und deren Eigenschaften und Methoden im nachhinein definiert. Eine einfachere Methode Objekte aufzubauen bieten *Objekt-Literale*. Durch eine Aufzählung von Schlüssel-Wert-Paaren (vgl. Listing 2.3) wird die Objektdefinition kompakt und übersichtlich. (vgl. [Stefanov, 2011, S. 42]) Objekt-Literale werden als *Konfigurations-Objekte* (vgl. [Stefanov, 2011, S. 80]) für den Datenaustausch und zur Umsetzung des *Singleton*⁹-Entwurfsmusters (vgl. Listing 2.3) eingesetzt. [Stefanov, 2011, S. 143]

Objekt-Erzeugung

```
1 var object = {
2   //Eigenschaft
3   property: "MyProperty",
4   //Getter-Methode
5   getProperty: function(){
6     return this.property;
7   },
8   //Setter-Methode
9   setProperty: function(property){
10    this.property = property;
11  }
12 };
```

Listing 2.3: Objekt-Literal mit Getter- und Setter-Methoden

⁹ Singleton ist ein Entwurfsmuster, das nur eine Instanz von einer Klasse erlaubt. Im klassenlosen JavaScript ist jedes erzeugte Objekt ein Singleton, da Objekte sich nicht gleichen. [Stefanov, 2011, S. 143]

Konstruktoren und Prototypen

Wie bereits erwähnt, kennt JavaScript keine Klassen, sondern nur Objekte. Dennoch lassen sich Konstruktoren definieren, die Instanzobjekte erzeugen. Ein Konstruktor ist in dem Fall nichts anderes als ein **Function**-Objekt. Jede Funktion besitzt eine Eigenschaft **prototype**, die bei der Initialisierung standardmäßig auf **Object.prototype** verweist. Durch *Prototypen* können Eigenschaften und Methoden auf alle Instanzobjekte, die mit der Funktion erzeugt wurden, vererbt werden. Das Vererbungsprinzip kann zum besseren Verständnis mit der Glasplattenmetapher nach [Debreuil, 2001] veranschaulicht werden. (siehe Abb. 2.6)

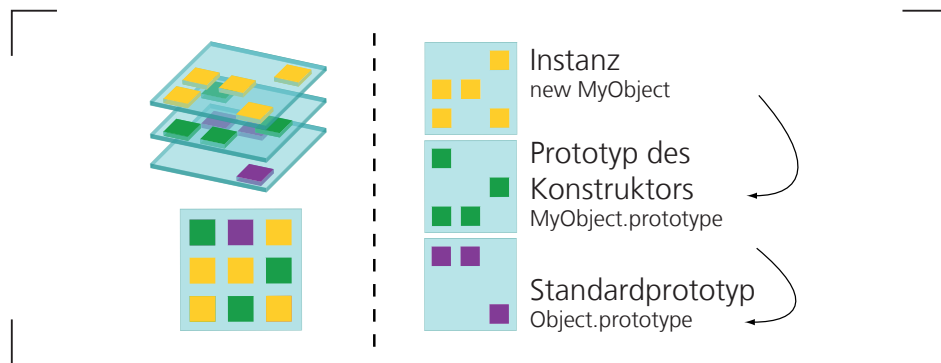


Abbildung 2.6: Verbildlichung des Vererbungsprinzips in JavaScript anhand der Glasplattenmetapher nach [Debreuil, 2001]

Prinzip der Vererbung durch Prototypen

Die Metapher beschreibt die Objekte in der sogenannten *Prototype-Chain* als Glasplatten. Auf ihnen sind jeweils Zettel angebracht, die eine Objekteigenschaft repräsentieren und deren Position ein Eigenschaftsname entspricht. Die Glasplatten werden übereinander gelegt. Auf unterster Ebene ist das initiale Prototyp-Objekt, darüber der Prototyp des Konstruktors und aufliegend die Instanz, die über den Konstruktor erzeugt wird. Schaut man nun durch alle Glasplatten, scheinen einige Eigenschaften der darunterliegenden Prototyp-Objekte durch. Andere werden wiederum von Instanz-Eigenschaften verdeckt. Der Zugriff erfolgt in jedem Fall - über die Instanz.

```
1 //Konstruktor-Definition
2 function MyObject() { .. }
3 //Erweiterung der Prototyp-Eigenschaften
4 MyObject.prototype.property = "Prototype";
5 //Erzeugung von Instanzen
6 var entity = new MyObject();
7 alert(entity.property); // "Prototype"
8 //Erweiterung der Instanz-Eigenschaften
9 entity.property = "Entity";
10 alert(entity.property); // "Entity"
```

Listing 2.4: Konstruktoren und Prototypen

In Listing 2.4 wird ein Beispiel der konkreten Implementierung in JavaScript gegeben. Es zeigt die Definition des Konstruktors mit dem Namen **MyObject** (Zeile 2) und der Objekt-Instanziierung (Zeile 6). In Zeile 4 wird eine Eigenschaft des Prototyp-Objektes überschrieben. Diese gilt für alle zugehörigen Instanzen (vgl. Zeile 7). Mit Zuweisung einer Instanz-Eigenschaft unter dem selben Eigenschaftsnamen (Zeile 9 und 10) wird die Prototyp-Eigenschaft überlagert. Dies gilt dann aber nur für die eine Instanz.

Prototypen geben die Möglichkeit Code wiederzuverwenden und performant zu halten, da Eigenschaften und Methoden nicht für jede Instanz neu erzeugt werden müssen. Weiterhin erlauben sie, bestehende Host- oder ECMAScript-Objekte um neue Funktionalität zu erweitern. Sie sind damit ein wichtiges Werkzeug für die Entwicklung anwendungsspezifischer Datenstrukturen.

*Mehrwert der
prototypbasierten
Programmierung*

Eine besondere Herausforderung ist die Kapselung von Skripten, um sie vor unerwünschten Zugriffen zu schützen. Ziel ist es, die Portierbarkeit in andere Umgebungen und Kompatibilität zu Code anderer Entwickler zu gewährleisten. Im Grunde sind dazu Namenskollisionen zu vermeiden. [Stefanov, 2011, S.11 ff.] Ein verbreitetes Muster ist das sogenannte *Revealing Module Pattern* [Schäfer, 2011a], das verschiedene Entwurfsmuster kombiniert. (vgl. [Stefanov, 2011, S.100 ff.]) Listing 2.5 zeigt dessen Umsetzung in JavaScript.

Module und Kapselung

```

1 // Erzeugung eines Namensraum
2 var NAMESPACE = {};
3 // Definition eines Moduls im Namensraum
4 NAMESPACE.Module = (function () {
5 // Private Eigenschaften und Methoden
6     var privateProperty = "private";
7     function privateMethod () { .. }
8 // Öffentliche Schnittstelle (API)
9     return {
10         publicProperty : "public",
11         publicMethod : function () { .. }
12     };
13 })();

```

Listing 2.5: Revealing Module Pattern

Um die Anzahl und Ausprägung globaler Variablen zu verringern und folglich Namenskollisionen zu umgehen, kann es sinnvoll sein Funktionen und Objekte in einem Objekt zu kapseln, das als Namensraum dient (Zeile 2). Ein Modul (Definition in Zeile 4) schließt zusammengehörige Teile der Funktionalität ein und repräsentiert eine abgeschlossene Komponente einer Software. Das Modul-Muster hilft neben der Kapselung dabei, den Code zu organisieren, indem es einen privaten Bereich (Zeile 5-8) und eine öffentliche Schnittstelle in einem `return`-Block (Zeile 9-12) definiert. Letztere ist von außen sichtbar. Private Eigenschaften und Methoden sind hingegen vor äußeren Zugriffen geschützt und können lediglich innerhalb des Moduls angesprochen werden.

Der Prozess der Gestenerkennung kann unter Umständen einige Zeit in Anspruch nehmen, was sich letztlich negativ auf das Bedienerlebnis¹⁰ auswirkt. Um die Performanz zu erhöhen, werden an kritischen Stellen der Verarbeitung oft Threads eingesetzt, die die Abarbeitung der Aufgaben parallelisieren. JavaScript bietet keine Threads an, dennoch lässt sich mit einem sogenannten *Web Worker*¹¹ ein ähnliches Verhalten erzielen. (vgl. [Stefanov, 2011, S. 192]) Dazu wird das rechenintensive Script in eine Datei ausgelagert, die von einem *Worker*-Objekt gesteuert wird. Die Kommunikation mit dem Hauptprogramm erfolgt über Events. Listing 2.6 zeigt die allgemeine Umsetzung eines Web Workers in JavaScript.

*Web Worker für die
parallele Verarbeitung*

¹⁰ Nach [Dorau, 2011, S. 20] ist die Performanz und Zuverlässigkeit eines Systems ein wichtiges Merkmal für ein positives Bedienerlebnis.

¹¹ Web Worker Spezifikation in [W3C, 2011h]

```
1 // Erzeugung des Worker-Objekts
2 var worker = new Worker('./path/to/myWorker.js');
3 // Registrierung auf Events des Workers
4 worker.onmessage = function(event){ .. };
5 // Kommunikation mit dem Worker-Script
6 worker.postMessage("Do something!");
```

Listing 2.6: Web Worker

Nachteilig ist jedoch die derzeit geringe Unterstützung des Konzepts. Insbesondere auf Plattformen mobiler Endgeräte ist eine Unterstützung erst in nächster Zeit abzu-sehen. Unter iOS sind Web Worker beispielsweise erst ab Version 5.0 implementiert. (vgl. [Deveria, 2011b])

2.3.6 Ereignisverarbeitung

*Bestandteile der
Ereignisbehandlung*

Wie in Kapitel 2.2.2 erläutert wurde, können Gesten durch ein Ereignisprofil be-schrieben werden. In diesem Zusammenhang erscheint es wichtig Grundlagen der Ereignisverarbeitung in JavaScript zu betrachten. Diese ist unter DOM Level 2 stan-dardisiert. [W3C, 2000] Darauf aufbauend werden Ereignisse überwacht, indem sie unter Angabe des Event-Typs auf einem Knoten des DOM-Baumes registriert wer-den. Als *Event-Handler* wird die Funktion bezeichnet, die unter dem Ereignis ausge-führt wird. Der Handler-Funktion wird dabei ein *Event-Objekt* als Parameter über-geben, welches nähere Informationen zu den Ereignisbedingungen liefert. Wie die Event-Schnittstelle folglich implementiert ist, zeigt Listing 2.7.

```
1 // Registrierung der Handler-Funktion auf einem Element
2 element.addEventListener("event",handler,capturing);
3 function handler(event){
4     alert(event.type); // Event-Typ
5 }
6 // Event-Handler entfernen
7 element.removeEventListener("event",handler,capturing);
8 // Ereignis auslösen
9 element.dispatchEvent(event);
```

Listing 2.7: API-Methoden für die Event-Registrierung nach DOM Level 2
[W3C, 2000]

Event-Typen

Der erste Parameter der Funktionen in Listing 2.7 legt den Event-Typ fest. Die folgen-de Auflistung führt die wesentlichen, durch eine Zustandsänderung im Eingabegerät initiierten, Event-Typen auf. Es handelt sich dabei um Maus- und Tastatureingaben, ergänzt um Ereignisse für die Auswertung der Orientierung mobiler Endegeräte. (vgl. [W3C, 2000], [W3C, 2011c] und [W3C, 2011b])

- Maus-Eingaben: `click`, `dblclick`¹², `mousedown`, `mouseup`, `mousemove`, `mouseover`, `mouseout`, `wheel`¹²
- Tastatur-Eingaben: `keyDown`¹², `keyUp`¹², `keyPress`¹²

¹² Eingeführt im Arbeitsentwurf der DOM Level 3 Event-Spezifikation in [W3C, 2011c]

- Orientierung¹³: **orientationchange**, **devicemotion**, **compassneedscale**

Es existieren weitere Ereignistypen, die sich speziell auf die Bedienelemente des HTML-Dokumentes (beispielsweise **focus**, **select** oder **submit**) oder des Browser-Interface (beispielsweise **resize** oder **scroll**) beziehen. Touch-Events werden gesondert an anderer Stelle dieser Arbeit betrachtet. (siehe Kapitel 3.2)

Der Parameter **capturing** der Interface-Methoden in Listing 2.7 ist ein boolescher Wert, der die Art der Event-Ausbreitung (oft auch als Event-Fluss bezeichnet) bestimmt. Dabei ist zu verstehen, dass ein abgesetztes Event einen Prozess, bestehend aus drei Phasen, durchläuft. Abbildung 2.7 veranschaulicht diese am Beispiel eines Events, das auf dem HTML-Element **div** definiert wurde. In der *Capturing*-Phase wird das Event im DOM-Baum von dem obersten Eltern-Knoten durch alle Element-Knoten auf dem Pfad zum Zielelement durchgereicht. Es folgt die *At-Target*-Phase, in der das Ereignis auf dem Zielelement, auf dem es registriert wurde, behandelt wird. Die *Bubbling*-Phase reicht das Event in entgegengesetzter Reihenfolge zum *Capturing*-Prozess weiter, sprich an alle Eltern-Elemente. Je nach Festlegung des Parameters kann entweder die *Capturing*-Phase oder die *Bubbling*-Phase übersprungen werden. Damit entscheidet sich der Entwickler für die Richtung des Event-Flusses. (vgl. [W3C, 2011c])

Die Weitergabe der Events kann nach Belieben mit der Methode **event.stopPropagation()** unterbunden werden. Überlicherweise implementieren die Web-Browser ein Standardverhalten für die Ereignisse. Beispielsweise wird bei der Selektion eines Eingabefeld ein Fokusrahmen als Markierung gesetzt. Auch wenn das primitive Feedback durchaus seine Berechtigung hat, kann es in speziellen Fällen stören. Mit der Methode **event.preventDefault()** lässt sich das Standardverhalten unterdrücken. [W3C, 2011c]

Event-Ausbreitung:
Capturing und Bubbling

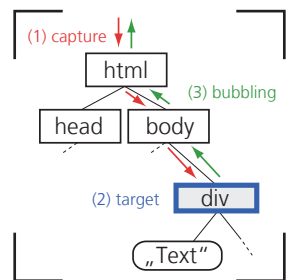


Abbildung 2.7: Phasen der Event-Ausbreitung im DOM-Baum

¹³ DeviceOrientation Event-Spezifikation [W3C, 2011b]

3 VERWANDTE ARBEITEN

3.1 ARCHITEKTURMODELL

Ziel der folgenden Betrachtung ist, den generellen Aufbau eines Gestenerkennungssystems und die Einordnung eines daraufbezogenen Frameworks, vor allem in Bezug auf den Einsatz in Webanwendungen, zu verstehen. Abbildung 3.1 zeigt den Ansatz einer allgemeinen Schichtenarchitektur nach [Echtler und Klinker, 2008] (links im Bild). In [Khandkar und Maurer, 2010a] und [Kammer et al., 2010] werden sprachbasierte Frameworks auf Basis dieses Architekturmodells vorgestellt. Bezüglich eines webbasierten Modells werden die Architekturschichten spezifiziert und in Abbildung 3.1 vergleichsweise dargestellt (rechts im Bild).

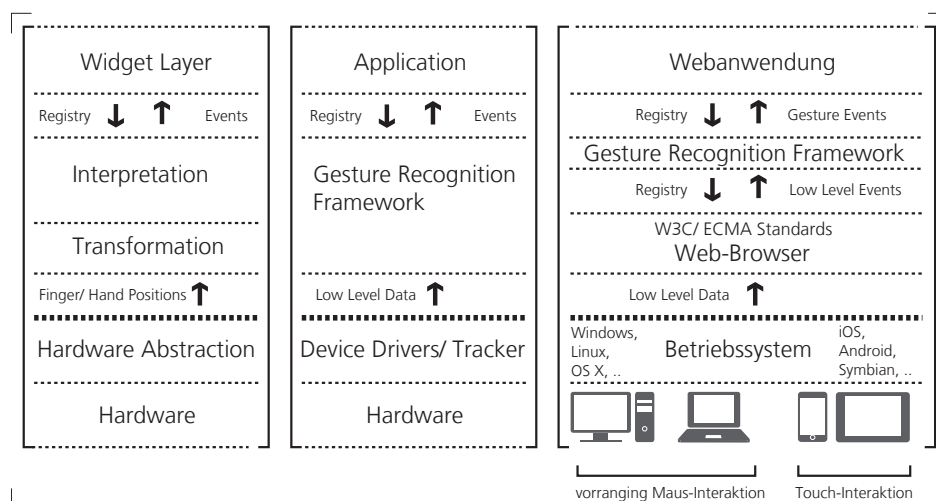


Abbildung 3.1: Architekturmodelle nach [Echtler und Klinker, 2008] (links), [Kammer et al., 2010] und [Khandkar und Maurer, 2010a] (mittig) gegenüber einem webbasierten Ansatz eines Gestenerkennungs-Frameworks (rechts)

Nach [Echtler und Klinker, 2008] gliedert sich ein Gestenerkennungs-Framework in vier Schichten. Indem Software-Komponenten der Eingabeverarbeitung entkoppelt werden, wird eine Modularisierung herbeigeführt, die letztlich die Erweiterungen und den Austausch von Komponenten ermöglicht. Der Ansatz ist für viele Vorbild, um eine geräteunabhängige und anwendungsunspezifische Implementierung eines Gestenerkennungs-Frameworks umzusetzen. (vgl. [Cheng et al., 2009], [Khandkar und Maurer, 2010a], [Kammer et al., 2010] und [Henzen, 2010])

*Allgemeine Architektur
nach ECHTLER ET AL.*

Hardware und Hardware-Abstraktion

Die *Hardware*-Ebene bildet die Basis eines jeden Eingabesystems. Bezüglich Webanwendungen kommen verschiedene Eingabegeräte zum Einsatz. Relevant sind dabei hauptsächlich die Eingabetechniken. Wie in Kapitel 2.2.1 differenziert, sind Maus-, Stift- und Touch-Eingaben zu unterscheiden. Die Interaktion mit der Hardware erzeugt Daten, die von der darüberliegenden Schicht *Hardware Abstraction Layer* in ein geräteunabhängiges Format übersetzt werden. Als Austauschformat zwischen Multitouch-Eingabengeräten und Verarbeitungsschichten wird häufig das *TUIO-Protokoll* verwendet (vgl. [Khandkar und Maurer, 2010a], [Görg, 2009], [Cheng et al., 2009] und [Henzen, 2010]). Das auf *OSC* (Open Sound Control) basierte Protokoll überliefert sowohl Berührungspunkte und deren Parameter, als auch physische Objekte, die auf der sensitiven Oberfläche erkannt werden (*Tangibles*). TUIO wird in Kapitel 3.2.2 nochmal aufgegriffen.

Transformation

Die Transformationsschicht dient nach [Echtler und Klinker, 2008, S. 2] vor allem der Integration des Kalibrierungsprozesses. So gesehen bildet die Schicht eine Programmierschnittstelle, die die Rohdaten nochmals filtert und bezüglich der Einstellungen in anschaulichere Daten übersetzt. Diese Abstraktion ist dem Web-Browser zuzuordnen, der seine APIs an Standards ausrichtet.

Interpretation und Applikation

In der darüberliegenden Schicht knüpft die Gestenerkennung an. Im Sinne einer losen Kopplung des Frameworks und der üblichen Verarbeitung von Nutzereingaben sieht das Modell eine eventbasierte Architektur vor. Auf Anwendungsebene erfolgt die Implementierung der konkreten Benutzeroberfläche, die sich hauptsächlich auf die grafische Visualisierung konzentriert. Die Umsetzung und Verarbeitung der gestenbasierten Interaktion der Web-Seite wird durch das Einbinden des Frameworks weitestgehend abgenommen.

Gliederung der folgenden Kapitel

Die vorgestellte Architektur einer webbasierten Gestenerkennungs-Software bildet die Grundlage für die weiteren Überlegungen bezüglich der Konzeption des Frameworks. Sie bestimmt die folgende Kapitelstruktur. Die Schicht der Hardware wird im Rahmen dieser Arbeit allerdings nicht näher beleuchtet. Für Informationen zu den Technologien berührungsempfindlicher Eingabegeräte sei an dieser Stelle auf [Teiche et al., 2009] und [Nygård, 2010] verwiesen. In Kapitel 3.2 wird der aktuelle Stand bezüglich der Abstraktion und Transformation der Eingabedaten erläutert, der sich in diesem Kontext auf den Webbrowser und dessen implementierte Web-Standards bezieht. Anschließend werden im Sinne der Interpretation von Gesten verschiedene Frameworks und Methoden der Gestenerkennung vorgestellt (siehe Kapitel 3.3). Schließlich werden in Kapitel 3.4 entsprechend der obersten Architektur-Schicht Aspekte des Interaktionsdesigns in Multitouch- und Webanwendungen thematisiert.

3.2 WEB-BROWSER

Die grundlegenden Funktionen eines Web-Browsers wurden bereits in Kapitel 2.1.2 beschrieben. Deren Implementierung variiert hinsichtlich der Erfordernisse des zugrunde liegenden Betriebssystems. Auf Basis der Empfehlungen des W3C, von deren Umsetzung auszugehen ist (vgl. Kapitel 2.1.3), kann das Web-Framework auf einheitlichen Schnittstellen aufgebaut werden. Die folgenden Abschnitte beschäftigen sich mit Ansätzen für die Einbindung elementarer, berührungsbasierter Eingaben in Webanwendungen.

3.2.1 Verarbeitung von Multitouch- und Stift-Eingaben

In Kapitel 2.3.6 wurden bereits Konzepte der Ereignisbehandlung in JavaScript beschrieben. Die Standardisierung bezüglich der Interaktion im Web bezieht sich dabei historisch begründet auf klassischen Eingabetechniken wie Maus- und Tastatureingaben. Auch wenn Browser, wie zum Beispiel der auf iOS basierende Safari [Apple, 2012], die Bedienbarkeit von Web-Seiten sicherstellt, indem er Gesten wie TAP auf Mausclicks technisch abbildet, entspricht dies nur einem Bruchteil des Potenzials der berührungssensitiven Interaktion. (vgl. Kapitel 2.2.3)

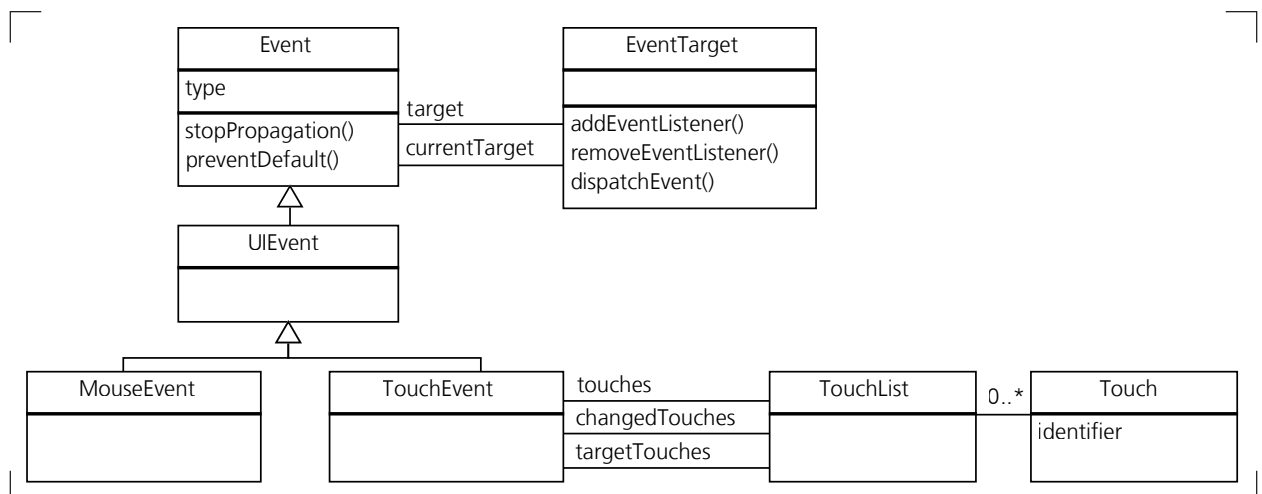


Abbildung 3.2: MouseEvent- und TouchEvent-Interface nach [W3C, 2000] und [W3C, 2011g]

Die *Web Events Working Group* des W3C arbeitet aktuell an einem Standard für die Integration von Multitouch- und Stifteingaben in Web-Seiten. Dessen Entwicklung schließt laut dem Meilensteinplan in [W3C, 2010b] mit einer offiziellen Empfehlung im August 2012 ab. Der aktuelle Arbeitsstand der Spezifikation ist in [W3C, 2011g] beschrieben. Demnach hat man sich auf folgende Event-Typen festgelegt: *touchstart*, *touchend*, *touchmove*, *touchcancel*, *touchenter*¹ und *touchleave*¹. Die Bedeutung der Ereignisse ist der von Maus-Events sehr ähnlich (vgl. Abschnitt 2.3.6). Die Unterschiede werden jedoch mit Betrachtung des Event-Interfaces, welches in Abbildung 3.2 dargestellt ist, deutlich.

*Standardisierung von
Multitouch- und
Stift-Eingaben*

¹Details der Spezifikation stehen noch zur Diskussion. Siehe W3C Editor's Draft der *Touch Events version 2* in [W3C, 2011f]

TouchEvent-Interface

TouchEvent erbt, wie auch das bekannte **MouseEvent** von dem Interface **UIEvent**, welches sich auf die vom Nutzer initiierten Ereignisse spezialisiert. Wesentliches Merkmal von Multitouch-Eingaben ist, dass mehrere Berührungseingaben synchron auftreten können. Das **TouchList**-Interface repräsentiert daher eine Menge Kontakte, welche jeweils durch den Datentyp **Touch** beschrieben werden. Jedem neuen Berührungspunkt wird dabei eine Nummer zugeteilt (**identifier**), wodurch eine Zuordnung in zuvor aufgetretenen oder darauffolgenden Ereignissen möglich ist. Es werden drei Typen von **TouchList** unterschieden. Das Attribut **touches** liefert eine Liste aller zum aktuellen Zeitpunkt auf dem Touchscreen registrierten Berührungspunkte. **changedTouches** beschränkt sich auf die Berührungspunkte, die sich zu den vorangegangenen Ereignissen verändert haben und **targetTouches** bezieht sich auf die Berührungspunkte, die dem Element zuzuordnen sind, auf dem die Geste begonnen wurde. **EventTarget** bildet die Schnittstelle für die Registrierung von Events, die bereits in Kapitel 2.3.6 erläutert wurde. (vgl. [W3C, 2000] und [W3C, 2011g])

Spezifikation von Eigenschaften eines Kontaktes

Je nach sensorischer Ausstattung des Gerätes, können verschiedene Informationen über einen Berührungspunkt erfasst werden. Folgende Eigenschaften eines Kontaktes sind im Interface **Touch** festgelegt:

- Position: **clientX**, **clientY**, **pageX**, **pageY**, **screenX** und **screenY**
- Druck: **force**
- Orientierung¹: **rotationAngle**
- Kontaktgröße¹: **radiusX** und **radiusY**

Umsetzung in Web-Browsern

Zahlreiche Web-Browser setzen die erläuterte Spezifikation, die sich maßgeblich an den Konzepten der *Safari Event API* orientiert (vgl. [Apple, 2011a]), bereits um. Die Unterschiede liegen oft im Detail, wie der Vergleich von Event-Schnittstellen verschiedener Plattformen und Web-Browser in [Koch, 2010], [Smus, 2011a] und [Blas, 2011] zeigt. Die Umsetzung einer Gesten-API schreitet zügig voran. Dies geht mit einem rasanten Prozess der Veröffentlichung und Veralterung von Browser-Versionen einher. Am Beispiel von Firefox wird dies deutlich (vgl. [Mozilla, 2011]): Mit der Version 3.5 (Juni 2009) wurden browserspezifische Mausgesten-Ereignisse wie zum Beispiel **MozSwipeGesture**, **MozMagnifyGesture** und **MozRotateGesture** umgesetzt, die mit der Maus oder einem Trackpad eingegeben werden. Version 3.6 (Januar 2010) erweitert die Browser-Funktionen mit dem **MozOrientation**-Event und spezifischen Media-Queries um Möglichkeiten, die Bewegungssensoren eines mobilen Endgerätes auszuwerten. Version 4 (März 2011) implementiert experimentell die Ereignisse **MozTouchDown**, **MozTouchUp** und **MozTouchMove** für die Unterstützung von Multi-Touch-Gesten. In Version 6 (August 2011) werden sie letztlich durch die erläuterte Spezifikation des W3C ersetzt. Mittlerweile wurde Version 10 herausgegeben. [Mozilla, 2012]

GestureEvent-Interface in Safari

Die Safari API implementiert eine weitere Schnittstelle. [Apple, 2011a] Das **GestureEvent**-Interface liefert Informationen über die geometrische Beziehung gleichzeitig registrierter Berührungspunkte. Das Attribut **scale** liefert beispielsweise den Abstand zwischen zwei Kontakten, **rotation** den Drehwinkel. Die Angaben sind relativ zur Ausgangssituation, die mit dem Start der Geste festgesetzt ist. Folgende Event-Typen können auftreten:

¹Details der Spezifikation stehen noch zur Diskussion. Siehe W3C Editor's Draft der *Touch Events version 2* in [W3C, 2011f]

- *gesturestart* - Tritt ein, sobald ein zweiter oder weiterer Finger den Touchscreen berührt.
- *gesturechange* - Tritt ein, wenn sich die Positionen der Kontakte innerhalb der Geste verändert.
- *gestureend* - Tritt ein, wenn die Geste endet, indem nur noch ein oder kein Finger den Touchscreen berühren.

3.2.2 TUIO-basierte Eingabeverarbeitung

Wie in Kapitel 3.1 eingeleitet wurde, hat sich das TUIO-Protokoll hinsichtlich einer geräteunabhängigen Implementierung einer Multitouch-Gestenerkennung durchgesetzt. Abbildung 3.3 verdeutlicht die allgemeine Funktionsweise eines TUIO-Frameworks nach [Teiche et al., 2009] und stellt ihr zwei Ansätze einer TUIO-Client-Implementierung für Webanwendungen nach [Rusadi, 2009] und [Cailliau et al., 2009] gegenüber. Die Eingabe-Hardware erzeugt zunächst einmal geräte- und technologiespezifische Sensordaten (*Raw Data*) wie zum Beispiel Bilddaten oder sonstige elektrische Messungen. Die Tracker-Implementierung wertet diese Datenströme zu diskreten Eingabedaten wie die Position oder Größe eines Kontaktes aus und bereitet diese im TUIO-Protokoll auf. Die genaue Spezifikation von TUIO kann in [Kaltenbrunner, 2011] nachgeschlagen werden. Der TUIO-Server bildet eine Vermittlungsschicht. Auf der Client-Seite empfängt ein installierter *TUIO Receiver* die TUIO-Pakete und gibt sie in adaptierter, aber nicht standardisierter Form an die Anwendung weiter. Es schließt sich je nach Anwendungskontext ein Gestenerkennungsmodul an.

TUIO
Framework-Architektur

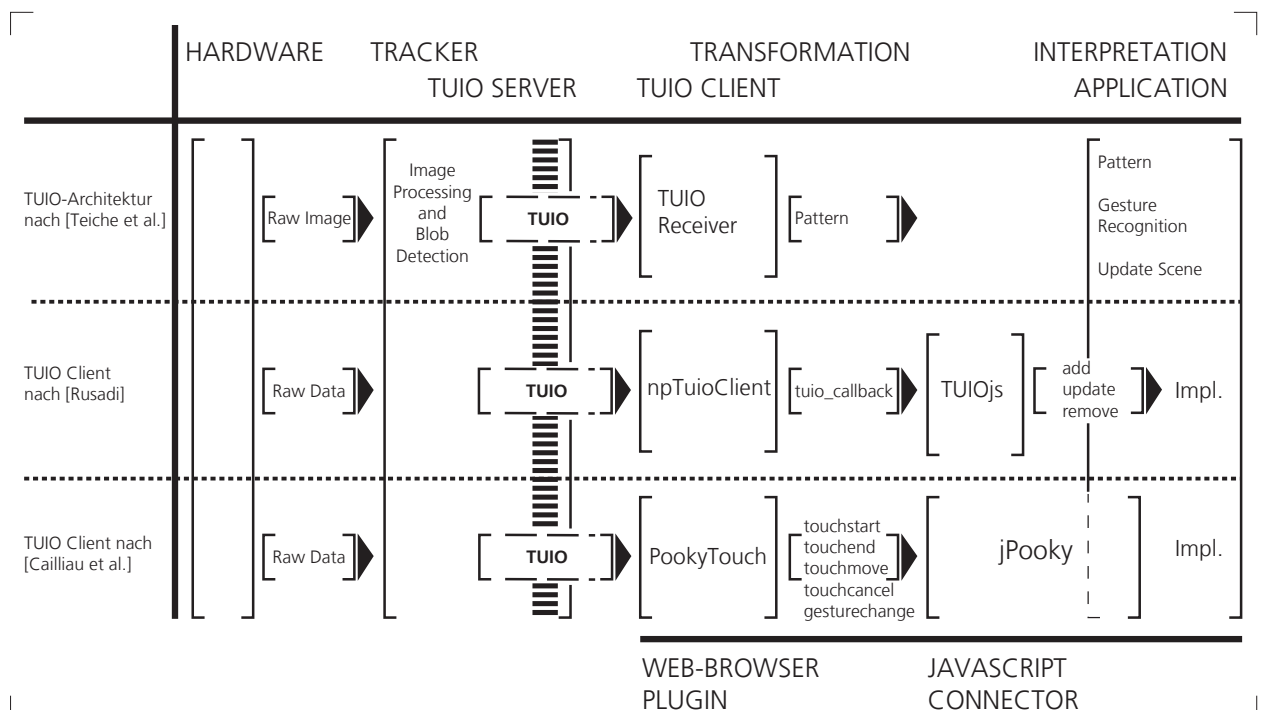


Abbildung 3.3: Implementierung eines webbasierten TUIO-Clients nach [Teiche et al., 2009], [Rusadi, 2009] und [Cailliau et al., 2009]

Ansätze der Client Implementierung

RUSADI und CAILLIAU ET AL. schlagen jeweils ein Browser-Plugin für die Aufgaben des *TUIO Receiver* vor. *npTuioClient* ist eine NPAPI²-basierte Implementierung eines Browser-Plugins und damit in verschiedenen Plattformen und Browsern wie Firefox, Chrome und Safari einsetzbar. Mit jedem Empfang von TUIO-Daten wird die JavaScript-Callback-Funktion `tuio_callback` aufgerufen. *TUIOjs* ist ein in JavaScript implementierter *Connector*, der die Daten ein weiteres Mal auswertet und Ereignisse wie *object_add*, *cursor_change* oder *cursor_remove* generiert. [Rusadi, 2009] In [Cailliau et al., 2009] hingegen werden die Ereignisse, die sich an der Safari API (vgl. [Apple, 2011a]) orientieren, bereits im Java-basierten Browser-Plugin *PookyTouch* erzeugt. *jPooky* ist als jQuery-Plugin implementiert, das die Ereignis-Handler definiert und damit ein festes Verhalten berührungsbasierte Manipulation von HTML- und SVG-Elementen festlegt. [Cailliau et al., 2009] Die JavaScript-Connector-Implementierung *MagicTouch* nach [Smus, 2011b] erweitert den Ansatz von RUSADI um standardkonforme Touch-Events, ähnlich wie sie in *PookyTouch* generiert werden. (vgl. [Smus, 2011b])

²NPAPI (Netscape Plugin Application Programming Interface) ist eine plattformübergreifende Plugin-Architektur, die von den meisten Web-Browsern unterstützt wird

3.3 INTERPRETATION VON GESTEN

3.3.1 Web-Frameworks

Im Zuge der Entwicklung der in Kapitel 3.2.1 beschriebenen Schnittstellen, entstanden auch zahlreiche Web-Frameworks, die die Integration von Multitouch-Gesten auf einer abstrakteren Ebene ermöglichen. Im Sinne der clientseitigen Web-Entwicklung basieren Web-Frameworks in der Regel auf der Programmiersprache JavaScript. Sie stellen einen problembezogenen Programmierrahmen bereit, der die Anwendungsentwicklung durch vereinfachte Programmierkonzepte erleichtert.

JavaScript-Framework

Einige namhafte und weit verbreitete Javascript-Frameworks sind *jQuery*, *Dojo Toolkit*, *MooTools*, *YUI Library* und *Ext JS*. (vgl. [Hallman, 2011]) Durch ihr Plugin-Konzept sind anwendungsspezifische Erweiterungen möglich, die auf ihrer Kernfunktionalität aufbauen. Für die Anwendung in touch-basierten mobilen Endgeräten wurden Erweiterungen entwickelt, mit denen sich bewährte Multitouch-Bedienkonzepte in mobilen Webanwendungen realisieren lassen. Dazu zählen die jQuery-Erweiterungen *jQTouch* [Kaneda und Stark] und *jQuery Mobile* [jQuery, 2011], das Dojo-Plugin *dojox.mobile* [Dojo, 2011], die Ext JS-Variante *Sencha Touch* [Sencha, 2011] und weitere Frameworks wie zum Beispiel *Qooxdoo* [Qooxdoo, 2011], *iUI* [Hewitt, 2011], *WebApp.NET* [Apers, 2011] oder *PastryKit* von Apple (vgl. [Calhoun, 2009]). Der Fokus dieser Implementierungen liegt auf der Nachbildung der Interaktionskonzepte und des Erscheinungsbildes nativer Anwendungen (*native Apps*³), sprich dem iOS- und Android-Interface. In den Abbildungen 3.4a-3.4c wird dies an ausgewählten Beispielen demonstriert. Das Gestenrepertoire beschränkt sich auf geläufige Standardgesten, die in der Regel an die vom Framework angebotenen User-Interface-Komponenten gebunden sind. Apple fasst die Anwendungen unter dem Begriff *WebApp* zusammen. (vgl. [Apple, 2011b])

Frameworks für mobile Webanwendungen

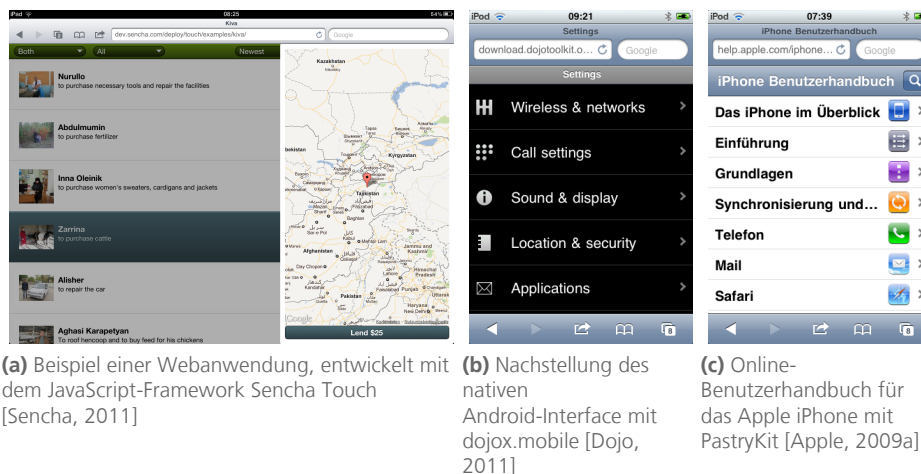


Abbildung 3.4: Showcases für die Verwendung von JavaScript-Frameworks für mobile Webanwendungen

³Eine *native App* ist eine plattformspezifische Anwendungssoftware in einem mobilen Endgerät wie beispielsweise einem Smartphone (vgl. [Charland und LeRoux, 2011])

Kompatibilität der Interaktionstechniken

Eine Herausforderung der geräteübergreifenden Darstellung von Web-Seiten ist die Bedienbarkeit unter verschiedenen Interaktionskonzepten zu gewährleisten. Dies erfordert die gleichbedeutende Unterstützung von Maus- und Berührungseingaben. Viele Web-Entwickler umgehen derartige Adaptionsprobleme, indem sie unterschiedliche gerätespezifische Versionen ihrer Anwendung implementieren. Es existieren jedoch Frameworks, die elementare Touch- und Mauseingaben weitestgehend aufeinander abbilden und vereinheitlichen. Die JavaScript-Bibliothek *Phantom Limb* [Carstensen, 2011] emuliert beispielsweise Touch-Events mit der Maus. Das jQuery-Plugin *Touchable* [Schwab, 2011] implementiert ebenso einen generischen Ansatz für die Verarbeitung von Maus- und Touch-Ereignissen.

Kontextunabhängige Lösungen der Integration von Gesten

Einige wenige Framework-Implementierungen konzentrieren sich allein auf die Erkennung von Gesten, wenn auch vorrangig Standard-Gesten, die an beliebige HTML-Elemente gebunden werden können. Ein Beispiel wurde in Kapitel 3.2.2 vorgestellt – *jPooky*. Auch *MooTools* bietet eine von grafischen Bibliotheken losgelöste Erweiterung [Pojer, 2011] für mobile Touch-Interfaces an. Im Folgenden wird die Verwendung der vergleichbaren unabhängigen Frameworks *WKTough* und *Jester* genauer beschrieben, um Erkenntnisse für eine geeignete Schnittstellenbeschreibung zu gewinnen.

WKTough

WKTough ist ein kleines Framework, mit dem die gestenbasierte Manipulation von HTML-Elementen ermöglicht wird. [Gibson, 2011] Das betrifft insbesondere das Skalieren, Rotieren und Verschieben von Objekten durch verbreitete Multitouch-Gesten. Für die softwaretechnische Umsetzung (siehe Listing 3.1) sieht das Framework das Objekt *WKTough* vor, mit dessen Instanz ein manipulierbares Objekt definiert wird. Der Objektbezug erfolgt ausschließlich über die Id des HTML-Elementes. Durch die Übergabe eines Konfigurationsobjektes, kann genau bestimmt werden, welche Operation und Eigenschaften auf dem Objekt möglich sein sollen.

```
1 var element = new WKTough('elementId', {
2   'draggable':true,
3   'scalable':false,
4   'rotatable': true,
5   'opacity':false
6 }).init();
```

Listing 3.1: Initialisierung von Multitouch-Gesten mit *WKTough* [Gibson, 2011]

Der Ansatz ist dahingehend interessant, dass Gesten implizit über zugewiesene Eigenschaften festgelegt werden, statt sie formal zu definieren und danach zuzuweisen. Darüberhinaus erfolgt die Objekterzeugung mit einhergehender Initialisierung der Gestenerkennung im Sinne des Entwurfsmusters *Immediate-Object-Initialisierung* nach [Stefanov, 2011, S. 76]. Das macht den Ansatz zu einer kompakten Lösung. Das Verhalten bei erfolgreicher Erkennung der Gesten ist wie in *jPooky* jedoch bereits vorgegeben. Eine Aktionszuweisung entfällt also.

Jester

Der Ansatz nach [Seaward, 2010] sieht in dem Framework *Jester* ebenfalls ein festes Repertoire möglicher Gesten vor, überlässt dem Web-Entwickler jedoch die Implementierung der Folgeaktion. *Jester* stellt acht verbreitete Gesten zur Verfügung. Sie werden jeweils durch Funktionen repräsentiert, denen gemäß einer eventbasierten Architektur ein Event-Handler übergeben wird. Zusätzlich bietet es Funktionen an, um elementare Ereignisse gemäß der drei Phasen (vgl. Kapitel 2.2.2) einer Geste zu verarbeiten.

Listing 3.2 verdeutlicht die Schnittstellen-Syntax. Die Funktion *jester* repräsentiert die Initialisierungs-Komponente und stellt die Objektbindung zu einem HTML-

Element her, das durch Zugriffsfunktionen der DOM-API zu erhalten ist. Der zweite Parameter bestimmt die Konfiguration der gebundenen Gesten. Im Beispiel wird die Interpretation einer TAP-Geste zeitlich begrenzt. Die Definitionen werden wie dargestellt funktional verkettet und beziehen sich damit allesamt auf die genannten Einstellungen. Die ausgewerteten Eingabedaten werden einer Handler-Funktion in einem benutzerdefinierten `TouchGroup`-Objekt geliefert, optional auch die Ausführungsrichtung.

```
1 jester(document.getElementById("elementId"), { tapTime: 80})
2   .tap(handler)
3   .doubletap(handler)
4   .swipe(handler);
5 var handler = function(touchGroup, direction) { .. }
```

Listing 3.2: Einbindung von Multitouch-Gesten in Jester [Seaward, 2010]

Moousture ist ein Framework, das auf der JavaScript-Bibliothek *Mootools* aufbaut. Das Ziel von *Moousture* lehnt an die Zielstellung dieser Arbeit an, die über die Verarbeitung von Standard-Gesten hinaus geht. Das Framework unterstützt die Entwicklung und Integration benutzerdefinierter Gesten und ermöglicht die Steuerung des Erkennungsprozesses (vgl. [Sibt-e-Hassan, 2010]). Die Komponenten des Frameworks beschränken sich im Grunde auf vier Klassen, die am Beispiel der Implementierung der Anwendungsschnittstelle in Listing 3.3 nachzuvollziehen sind.

```
1 // Erzeugung des Gestenerkenners
2 var gesture= new Moousture.ReducedLevenMatcher();
3 // Beschreibung und Erzeugung der Geste
4 gesture.addGesture([3,2,1,0,7,6,5,4], ccwCircle);
5 // Event-Handler
6 function ccwCircle(error){ .. }
7 // Objekt-Zuweisung
8 var probe = new Moousture.MouseProbe($(document));
9 // Konfiguration
10 var recorder = new Moousture.Recorder({
11   maxSteps: 20,
12   minSteps: 8,
13   matcher: gesture
14 });
15 // Steuerung des Erkennungsprozess
16 var monitor = new Moousture.Monitor(30, 2);
17 monitor.start(probe, recorder);
18 monitor.stop();
```

Listing 3.3: Formalisierung und Registrierung einer Kreis-Geste in *Moousture* [Sibt-e-Hassan, 2010]

Zunächst wird der Gestenerkennner initialisiert und mit ihm der Erkennungsalgorithmus festgelegt. (vgl. Listing 3.3 Zeile 2) Eine weitere Aufgabe dieser Komponente, ist die Registrierung der Geste. Die Methode `addGesture` nimmt hierfür zwei Parameter: Eine Repräsentation der Geste und eine Handler-Funktion für die Aktions-Zuweisung. Der Ansatz ist, den Pfad einer Einzelpfad-Geste als eine Sequenz formaler Bewegungsrichtungen zu beschreiben. Abgebildet auf acht diskrete Werte (siehe Abb. 3.5), wird dem Gestenerkennner demnach eine Liste von Zahlen übergeben. Im Code-Beispiel repräsentiert die Sequenz einen Kreis, der gegen den Uhrzeigersinn

*Benutzerdefinierte
Gesten mit Moousture*

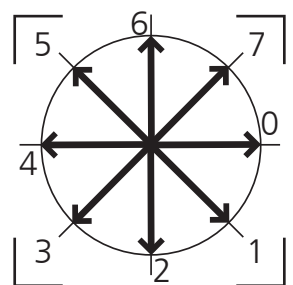


Abbildung 3.5: Abbildung der Himmelsrichtungen nach [Sibt-e-Hassan, 2010]

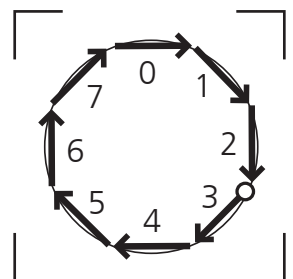


Abbildung 3.6: Formale Beschreibung einer Kreisbewegung nach [Sibt-e-Hassan, 2010]

ausgeführt wird. (vgl. Abb. 3.6) Die Zuweisung der Geste zu einem Geltungsbereich, wird mit der Klasse **Moosture.MouseProbe** realisiert. (vgl. Listing 3.3 Zeile 8) Entsprechend der Bezeichnung wird in dem Fall eine Mausgeste behandelt. Es existiert jedoch auch eine Erweiterung für den Einsatz in touchbasierten Webanwendungen, die durch die Funktion **Moosture.iPhoneProbe** gekapselt ist. Das Objekt, auf dem die Geste definiert ist, wird mit der in Mootools integrierten Selector-Engine **\$(element)** ausgewählt und der Klasse übergeben. Den Verarbeitungsprozess der Nutzereingaben übernimmt die Klasse **Moosture.Recorder** auf Basis der gegebenen Konfiguration. (vgl. Listing 3.3 Zeile 10) Zuletzt bestimmt die Komponente **Moosture.Monitor** den Ablauf der Gestenerkennung. (vgl. Listing 3.3 Zeile 16) Die initialen Parameter legen die Genauigkeit und Fehlertoleranzgrenze fest. Die Erkennungsphase wird letztlich mit den Methoden **start** und **stop** gesteuert.

3.3.1.1 Zusammenfassung

Die Recherche zu webbasierten Frameworks, die die Entwicklung von Gesten auf Multitouch-Systemen unterstützen, zeigt ein breites Spektrum an Lösungen auf. Nur wenige zielen jedoch auf eine Konzeption anwendungsspezifischer Gesten ab. Mit Moosture wurde ein Framework gefunden, das eine Formalisierung von Gesten in Betracht zieht und damit den Anforderungen an diese Arbeit nahe kommt, allerdings können hiermit lediglich Einzelpfad-Gesten angewandt werden. Es wurde festgestellt, dass sich eine Vielzahl von Frameworks auf die Imitierung von plattformspezifischen Benutzeroberflächen konzentrieren (*WebApps*). CHARLAND ET AL. sind der Auffassung, dass derartige Implementierungen die Zukunft mobiler Software sind: *"We're confident that Web technologies will become indistinguishable from native experiences"*. [Charland und LeRoux, 2011] Entgegen dieser Aussage wird in dieser Arbeit jedoch der Standpunkt vertreten, dass die Frameworks die gestalterischen Freiheiten eines Webentwicklers sehr stark einschränken, indem sie Bedienelemente und darauf anwendbare Gesten vorgeben. Das Fazit dieses Kapitels ist demnach, dass trotz vieler Lösungen nach wie vor ein Mangel an Instrumenten für die Implementierung touchbasierter Web-Interfaces besteht, insbesondere in Hinsicht auf die Konzeption selbst definierter Multitouch-Gesten.

3.3.2 Sprachbasierte Frameworks

Die Implementierung spezieller anwendungsspezifischer Gesten auf Basis einer universell einsetzbaren Programmiersprache wie JavaScript erfordert fundiertes Wissen der Softwareentwicklung und Kenntnisse über die Verwendung des zu Grunde liegenden Event-Interface. Die Abbildung der Komplexität und Vielfalt von Gesten kann unter Umständen sehr kompliziert und unübersichtlich werden. Aus diesem Grund werden in diesem Kapitel Ansätze der Gestenformalisierung betrachtet.

Klassifikation von Formalisierungsansätzen

Eine umfassende Einordnung von Ansätzen der Gestenbeschreibung wird in [Henzen, 2010] vorgenommen. Demnach werden *lineare*, *baumstrukturierte* und *netzartige* Formalisierungsansätze unterschieden. Darüber hinaus kann eine Gestenbeschreibung bezüglich ihrer Darstellungsart und dessen *Formalisierungsgrad* (vgl. [Kammer et al., 2011]) klassifiziert werden. Eine für den Menschen intuitiv verständliche Methode ist die natürliche Sprache und bildliche Verdeutlichung (*Benutzerstufe*, vgl. [Kammer et al., 2011]), beispielsweise durch die Skizzierung der Gestenpfade. Wie in [Henzen, 2010] bemerkt wird, ergeben sich hinsichtlich einer rechnergestützten

Auswertung jedoch Probleme, da sich Mehrdeutigkeiten ergeben können. Formale Sprachen hingegen basieren auf einer strukturierten und logischen Verknüpfung von Zeichen, die mit Kenntnis ihrer Semantik, sowohl aus Benutzersicht als auch von Rechensystemen lesbar und auswertbar sind. In [Kammer et al., 2011] wird dies folglich als *Austauschstufe* bezeichnet. KHANDKAR ET AL. bringen den Zweck einer formalen Gestenbeschreibungssprache auf den Punkt:

“A domain-specific language (DSL)⁴ for definition gestures can hide the low level complexities by encapsulating complex mathematical calculations, pattern recognition algorithms and the like. This can help the developers focus on designing the gesture at a level that is most appropriate to the application context without worrying about the implementation details.” [Khandkar und Maurer, 2010a]

Während die Sprachgrammatik das Format der Verständigung festlegt, ist auf Systemebene eine Auswertung der erkannten Eingabeparameter gegenüber den formal beschriebenen Attributen vorzunehmen. Dies erfordert eine Übereinkunft auf mathematischer Ebene, die durch die Modellierung interner Datenstrukturen herbeigeführt wird. Dieser Formalisierungsgrad wird in [Kammer et al., 2011] auch als *Algorithmenstufe* bezeichnet.

Die Implementierung der genannten Formalisierungsgrade und ihre Verknüpfung in einem Framework führt nicht nur Abstraktionsebenen ein, die dem Entwickler ein Ausdrucksmittel in die Hand geben, um gestenbasierte Interaktionskonzepte in seiner Anwendung effizienter zu entwickeln, sondern erzeugt auch eine Trennung von Verantwortlichkeiten in der Frameworkarchitektur. Die modulare Umsetzung führt nicht zuletzt zu einem höheren Maß an Wiederverwendbarkeit, Austauschbarkeit und Erweiterbarkeit von Komponenten der Gestenerkennung. Diese Zusammenfassung entspricht im wesentlichen den Anforderungskriterien der in [Henzen, 2010] vorgestellten sprachbasierten Frameworks. Darunter das auf einer mathematisch geprägten Gestenbeschreibung basierende *FRAMOUS* (Framework for abstract representation of Multitouch gestures) [Görg, 2009] und der generische Ansatz von *libTisch* (Tangible Interactive Surfaces for Collaboration between Humans) [Echtler und Klinker, 2008], auf dem die formale Merkmalspezifikation von Gesten nach [Echtler et al., 2010] aufbaut.

*Ableitung der
Frameworkarchitektur*

Erweiternd werden im Folgenden zwei Frameworks auf Basis regelbasierter Sprachansätze nach [Khandkar und Maurer, 2010a] und [Scholliers et al., 2011] vorgestellt. Anschließend wird in Abschnitt 3.3.2.3 die Beschreibungssprache *GeForMT*, die den Schwerpunkt dieser Arbeit ausmacht, näher gebracht.

*Gliederung der
folgenden Abschnitte*

3.3.2.1 TouchToolkit

TouchToolkit ist ein Softwareentwicklungswerkzeug, das die Entwicklung und den Test von Multitouch-Anwendungen auf Basis der Microsoft-Technologien *Windows Presentation Foundation* (WPF) und *Silverlight* unterstützen soll. [Khandkar und Maurer, 2010b] Abbildung 3.7 zeigt die Frameworkarchitektur, deren Grundzüge bereits in Kapitel 3.1 im Vergleich zu [Echtler und Klinker, 2008] und [Kammer et al., 2010]

Prinzipieller Aufbau

⁴DSL (Domain Specific Language - z. dt.: Domänenspezifische Sprache) ist eine formale Sprache, die für ein spezifisches Problemfeld entworfen ist und eine fachlich ausgerichtete, leichter erlernbare Programmierschnittstelle bildet.

dargelegt wurden. Die Definition von Gesten erfolgt auf Basis der *Gesture Definition Language* (GDL) nach [Khandkar und Maurer, 2010a], die als Konfiguration des Frameworks ausgelagert ist. Jede Definition beschreibt eine Geste, die über Registrierfunktionen der API zur Laufzeit eingebunden werden können. Die Regeln werden in ein internes Datenmodell transformiert, auf dessen Basis der Gestenprozessor eine Validierung der Eingabedaten vornimmt.

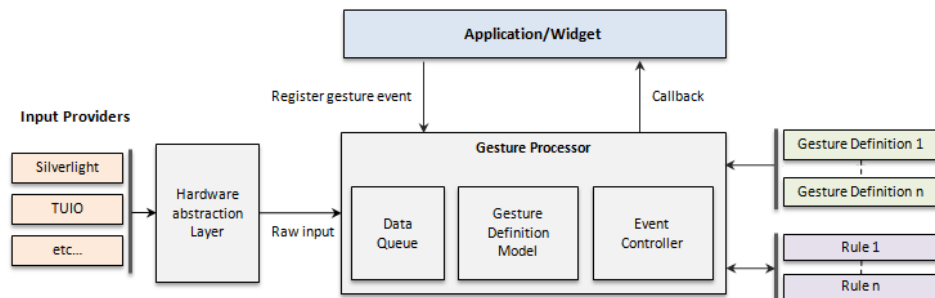


Abbildung 3.7: Komponenten in TouchToolkit [Khandkar und Maurer, 2010b]

Gestendefinition in GDL



Abbildung 3.8:
LASSO-Geste zur
Auswahl einer
Objektmenge
[Khandkar und Maurer,
2010a]

Eine GDL-Definition beginnt mit dem Schlüsselwort **name**, gefolgt von einem eindeutigen Bezeichner der Geste. Einige häufig verwendete Gesten sind zur Wiederverwendung bereits vordefiniert (vgl. [Khandkar und Maurer, 2010b]). Eine Überschreibung zum Zweck der Neudefinition dieser Standardgesten ist unter Angabe des selben Bezeichners möglich. Mit dem Schlüsselwort **validate** wird der Block für die Regeln, die die Geste beschreiben, eingeleitet. Sie setzen sich aus atomaren Bedingungen zusammen. Das Beispiel in Listing 3.4 definiert eine LASSO-Geste (vgl. Abb. 3.8) mit folgenden Einschränkungen: Die Eingabe wird mit einem Finger ausgeführt (**Touch limit**), formt einen geschlossenen Kreis (**Closed loop**) und wird ausgewertet, sobald der Finger von der sensitiven Oberfläche abgehoben wird (**Touch state**). Darüber hinaus werden die Dimensionen bezüglich des Geltungsbereichs der Geste (**Touch path bounding box**), der Pfadlänge (**Touch path length**) und der eingeschlossenen Fläche (**Enclosed area**) festgelegt. Der **return**-Block beschreibt das Ergebnis, das zurückgeliefert wird, wenn die Eingabe alle Bedingungen der Geste erfüllt. Dabei kann genaustens bestimmt werden, welche Information für die jeweilige Geste in der Rückgabe relevant sind.

```
name: Lasso
validate
  Touch state: TouchUp
  Touch limit: 1
  Closed loop and
  Touch path bounding box: 200x200..1000x1000 and
  Touch path length: 600..100000 and
  Enclosed area: 5000..1000000
return
  Touch points
```

Listing 3.4: LASSO-Geste in GDL [Khandkar und Maurer, 2010a]

Kombination und Erweiterung der Sprachoptionen

Die Möglichkeit mehrere **validate**-Blöcke in einer Definition zu definieren verspricht Flexibilität in der Beschreibung komplexer Gesten. Die Regelblöcke können jeweils adressiert werden, um sie in einer Kompositionsregel wiederzuverwenden. Auch bezüglich der Erweiterung der Sprache geht das Framework mit gutem Beispiel voran und bietet eine Schnittstelle in GDL an, um neue Bedingungenregeln hinzuzufügen, die in den Gestendefinitionen verwendet werden können. (vgl. [Khandkar und Maurer, 2010a])

3.3.2.2 Midas

Eine ähnliche Konstruktion von Regeln zur Modellierung von Multitouch-Gesten, mit vergleichbaren Zielen wie GDL, erdachten sich SCHOLLIERS ET AL. in dem Framework *Midas*. Die Formalisierung sieht hingegen einzelne Events oder Event-Sequenzen als Basismodell vor, auf dem die Regeln angewendet werden.

Abbildung 3.9 zeigt den Aufbau des Frameworks. Die Basis der Eingabeverarbeitung bildet das *Infrastructure Layer*, in dem Eingabeströme adaptiert und in logische Fakten (*Fact Base*) übersetzt werden. Im *Core Layer* werden sie mit einer Regelmeng (Rule Base) abgeglichen, die zuvor im *Application Layer* definiert wurde. Diese Vergleichslogik der *Inference Engine* produziert weitere Fakten für die Interpretation der Eingabe und löst Prozeduren auf Anwendungsebene aus.

Architektur

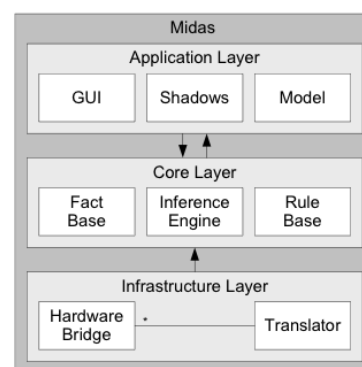


Abbildung 3.9: Midas Framework-Architektur [Scholliers et al., 2011]

Eingeleitet wird eine Regel mit dem Schlüsselwort **defrule** und dem Bezeichner der Geste. Listing 3.5 verdeutlicht die Syntax am Beispiel einer FLICK-Geste. Jede Eingabe produziert einen Fakt vom Typ **Cursor**, beschrieben durch eine Id, der Position, dem Status des Ereignis, sowie räumlichen und zeitlichen Parametern. Mit der **ListOf**-Operation können Bedingungen über eine Menge von **Cursor**-Ereignissen festgelegt werden. Das Beispiel definiert eine Sequenz von Ereignissen derselben Finger-ID im Zeitrahmen von 500 Millisekunden, die mindestens fünf **Cursor**-Events beinhaltet. Für Relationen bezüglich der Zeit, Dimension und Bewegung einer Geste existieren entsprechende Operationen. Die Operation **movingLeft** bedingt beispielsweise eine links ausgerichtete Bewegung. Möglich sind auch **movingRight**, **movingUp** und **movingDown**. Analog können Operationen wie **tEqual**, **tMeets**, **tBefore**, **tAfter** und **tContains** für temporale Relationen und **sDistance**, **sNear**, **sNearLeftOf**, **sNearRightOf** und **sInside** für räumliche Relationen angewandt werden. Sind alle definierten Bedingungen der Regel erfüllt, wird eine Aktion ausgelöst, die nachfolgend mit dem vorangestellten Zeichen **=>** beschrieben ist.

Regel-Syntax

```

(defrule FlickLeft
  ?eventList[] <- (ListOf
    (Cursor (same: id) (within: 500) (min: 5)))
  (movingLeft ?eventList)
  => (assert (FlickLeft (events ?eventList))))

```

Listing 3.5: Aufbau einer Regel am Beispiel der FLICK-Geste [Scholliers et al., 2011]

Objektbezug und
Komposition

Während TouchToolkit den Objektbezug außerhalb der Beschreibungssprache herstellt (vgl. [Khandkar und Maurer, 2010a]), konstruiert Midas sogenannte *shadow facts*, die in der Definition angewendet werden können. *Shadow facts* bilden die Attribute und Methoden entsprechend der Klassendefinition eines grafischen Elements ab und erlauben so einen transparenten Zugriff auf deren Eigenschaften innerhalb der Definition. Das Regelwerk ist sehr umfassend und bietet viele Ausdrucksmöglichkeiten an. Dem kommt die Wiederverwendung einer Geste über ihren Bezeichner entgegen, die in komplexeren Definitionen verbaut werden kann.

3.3.2.3 GeForMT - Gestenformalisierung für Multi-Touch

Sprachbausteine nach
KAMMER ET AL.

KAMMER ET AL. schlagen mit *GeForMT* (Gesture Formalization for Multi-touch) eine Beschreibungssprache für Multitouch-Gesten vor, die nach [Kammer et al., 2010] im Wesentlichen aus fünf Sprachbausteine besteht. Die *Funktion* beschreibt die Art der Eingabe und Anzahl simultaner Kontakte einer Geste (beispielsweise *2F(..)*, vgl. Abb. 3.10a). *Atomare Gesten* gelten als die elementaren Einheiten einer Bewegung. Sie bilden sowohl punkthafte Gesten wie *POINT* und *HOLD*, als auch eindimensionale (*LINE*) und zweidimensionale Pfade (*MOVE*, *CIRCLE*, *SEMICIRCLE*) ab. Zusätzliche Richtungsangaben in Form einer Himmelsrichtung (beispielsweise *NORTH* oder *SOUTHEAST*) oder Kreisbewegungsrichtung (beispielsweise *CLOCKWISE*) konkretisieren die Definition der Bewegungsform (vgl. Abb. 3.10b). *Operationen* ermöglichen die Komposition atomarer Gesten zu komplexen Definitionen. So kann beispielsweise durch eine Sequenz atomarer Gesten, verknüpft durch den Komma-Operator, ein komplexes Symbol gebildet werden (vgl. Abb. 3.10c) oder durch die Verknüpfung mit dem Stern-Operator eine zeitgleiche Ausführung von Bewegungen definiert werden (vgl. Abb. 3.10d). Die Komposition von Gesten-Atomen lässt viele Variationen zu. Durch *Relationen* werden genauere Bedingungen bezüglich der Beziehung verknüpfter Gesten festgelegt. Der Ausdruck *CROSS* bedingt beispielsweise, dass sich die Pfade der Gesten überschneiden (Abb. 3.10c). *JOIN* und *SPLIT* fordern dass sich die Kontakte aufeinander zu oder voneinander weg bewegen (Abb. 3.10d). Der *Fokus* beschreibt den Geltungsbereich, auf dem eine Geste definiert wird. Er kann sich über ein einzelnes Objekt, eine Objektmenge oder die gesamte Benutzeroberfläche erstrecken.

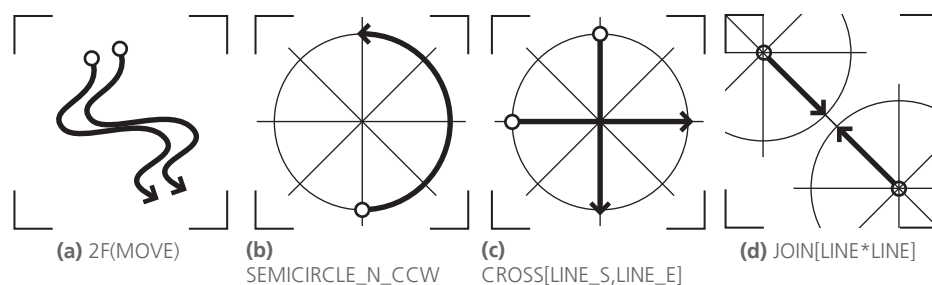


Abbildung 3.10: Beispiele der Formalisierung mit GeForMT

Gestenerkennungs-
Framework basierend
auf GeForMT

Die Entwicklungsarbeit an dem Gestenerkennungs-Framework *GeRahMT* (GeForMT Rahmenwerk für Multitouch) in [Henzen, 2010] deckt Schwachstellen der Formalisierung auf, aus denen entsprechende Spracherweiterungen resultieren. Der aktuelle Stand der GeForMT-Syntax stellt sich in Listing 3.6 entsprechend dar. Aufgenommene Sprachzusätze nach [Henzen, 2010] betreffen hauptsächlich die Angabe eines Bezeichners durch die EBNF-Regel *name*, Variationen von Gesten durch Optionen (*Oder-Operator* 'I') und tendenziellen Richtungsangaben (beispielsweise *N°*), eine präzisere Lagebeschreibung durch die Relation *CONNECT* und Parametern für

die Art der Gestenerkennung (ON und OFF). Die Wiederverwendung des Bezeichners bereits definierter Gesten, wie es in TouchToolkit und Midas möglich ist, wird hier ebenso in Betracht gezogen. (vgl. [Kammer et al., 2011])

```

definition ::= name '=' complex('|'complex)* [' ':'time'];
name       ::= ([a-z][A-Z])([a-z][A-Z])*
complex    ::= gesture (operator gesture)*
              | relation '[' gesture (operator gesture)* ']'
gesture    ::= function '(' atomfocus(',', atomfocus)* ')'
              | atomfocus(',', atomfocus)*
atomfocus  ::= atom | atom('focuslist')
focuslist  ::= focus(',', focus)*
focus     ::= 'a'|'s'|'o'[integer]lm[integer]
function    ::= [integer] 'F' | [integer] 'H' | [integer] 'B'
operator    ::= '*' | '+' | ',' | '|'
relation    ::= 'SYNC'|'JOIN'|'SPLIT'|'CROSS'|'CONNECT'
atom        ::= identifier
              | vector['_'direction]
              | shape['_'direction']['_'rotation]
identifier  ::= 'MOVE' | 'POINT' | 'DEPOINT' | 'HOLD'
vector      ::= 'LINE'
shape       ::= 'CIRCLE' | 'SEMICIRCLE'
direction   ::= 'NORTH'|'EAST'|'SOUTH'|'WEST'
              | 'N'|'E'|'S'|'W'
              | 'NORTHEAST'|'SOUTHEAST'|'SOUTHWEST'|'NORTHWEST'
              | 'NE'|'SE'|'SW'|'NW'
              | 'N°'|'E°'|'S°'|'W°'
rotation    ::= 'CLOCKWISE'|'CW'|'COUNTERCLOCKWISE'|'CCW'
integer     ::= [1-9][0-9]*
time        ::= 'ON' | 'OFF'

```

Listing 3.6: GeForMT-Syntax-Spezifikation in EBNF [Kammer et al., 2011]

Die Beziehung zu Objekten der Benutzeroberfläche wird in der *focus*-Regel hergestellt, unterliegt jedoch einer eher unscharfen, abstrahierten Klassifikation. Darüber hinaus beinhaltet die Sprache im Vergleich zu TouchToolkit und Midas keine Ausdrucksmöglichkeiten über die Rückkopplung, also welche Informationen die Applikation erhalten soll und wie sie sie entgegen nimmt. Der Entwurf dieser Schnittstellen unterliegt plattformspezifischen Entwurfsmustern, weshalb eine konkretere Formulierung an Allgemeingültigkeit verlieren würde. HENZEN beschreibt die Interaktionskomponenten *Geste*, *Objekt* und *Aktion* als Dreieckskonstellation (vgl. Abb. 3.11), die sich auf der Grundlage des Implementierungswissens formal abbilden und verknüpfen lassen. Der Ansatz verfolgt eine klassenbasierte Repräsentation von Objekten, in der der Klassenname und zugehörige Methoden in der Formalisierung verwendet werden, ähnlich wie *Shadows* in Midas.

*Schnittstellen zwischen
Formalisierung und
Implementierung*

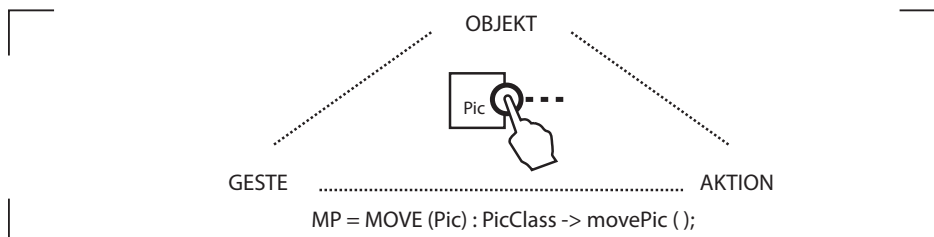


Abbildung 3.11: Formalisierung des Objekt- und Aktionsbezugs [Henzen, 2010]

3.3.2.4 Zusammenfassung

Gegenüber den Formalisierungsansätzen in TouchToolkit und Midas konzentriert sich GeForMT auf die Beschreibung der wesentlichen Bewegungsform einer Geste. Die weitgehende Selbstbeschreibungsfähigkeit der Syntax und ihr einfacher Aufbau, der an mathematische Ausdrücke erinnert, erhöht nicht nur die Lesbarkeit und senkt die Lernkurve, sondern ermöglicht bereits in einem einzeiligen Ausdruck die Spezifizierung einer komplexen Gesten. Die Gestendefinition in Midas setzt durch ihr eventbasiertes Konzept deutlich mehr Implementierungswissen voraus und ist dadurch verhältnismäßig kompliziert. Die eigentümliche Syntax der Variablen- und Funktionsdefinitionen, sowie deren Klammerung erschweren die Anwendung auf erdachte Gesten. Die Annäherung an programmierbare Datenstrukturen erhöht dafür aber den Abstraktionsgrad des Erkennungsalgorithmus und lässt somit Erweiterungen über die Sprache zu. TouchToolkit strukturiert die Definition in Blöcken und fördert dadurch die Übersichtlichkeit. Wie in Midas werden die Gesten aufbauend auf harten Fakten spezifiziert. Im Gegensatz zu den geometrischen Primitiven in GeForMT, deren räumliche Verhältnisse und temporalen Eigenschaften unbestimmt und damit variabel sind, können die gültige Wertebereiche für die Erkennung einer Geste sehr genau eingestellt werden.

3.3.3 Methoden der Gestenerkennung

Wie im Zusammenhang mit der Beschreibung von Formalisierungsgraden im letzten Kapitel erklärt wurde, erfordert die Erkennung formal definierter Gesten repräsentative, mathematisch berechenbare Datenmodelle. Im Folgenden werden verschiedene Ansätze der Gestenerkennung beschrieben, die anschließend für die Verwendung in einem Web-Framework bewertet werden.

Hidden Markov Model

Eine verbreitete Methode in der Mustererkennung ist das *Hidden Markov Model* (HMM). Es setzt sich aus zwei stochastischen Prozessen zusammen. Der erste Prozess wird durch eine *Markow-Kette* bestehend aus Zuständen und Zustandsübergängen beschrieben. Eine Markow-Kette zeichnet sich dadurch aus, dass die Wahrscheinlichkeit eines Übergangs zu einem bestimmten Zustand allein von dem Vorgänger-Zustand abhängt. Ein zweiter Zufallsprozess erzeugt auf dieser Grundlage zu jedem Zeitpunkt ein beobachtbares Ausgabesymbol. (vgl. [Wood, 2008]) Die Wahrscheinlichkeit mit der eine Ausgabesequenz erzeugt wurde, kann mit dem sogenannten *Vorwärts-Rückwärts-Algorithmus* ermittelt werden. [Wood, 2008] Die Zustände der Markow-Kette bleiben dabei verborgen (*hidden*). Um die Parameter zu ermitteln, die die wahrscheinlichste Ausgabesequenz erzeugen, wird der *Baum-Welch-Algorithmus* angewendet. [Damaraju und Kerne, 2008] Die Modellparameter werden daraufhin optimiert. Eine beispielhafte Umsetzung des HMM wird in [Webel et al., 2008] vorgestellt. WEBEL ET AL. schlagen eine HMM-basierten Programmierschnittstelle vor, die den X3D⁵-Standard um Knoten für die Multitouch- Gestenerkennung erweitert. Um Gesten korrekt klassifizieren zu können, müssen sie jedoch zunächst aufwendig antrainiert werden (vgl. [Webel et al., 2008] und [Damaraju und Kerne, 2008]).

⁵X3D: Ein XML-basiertes Standardformat zur Beschreibung von 3D-Modellen, die im Webbrowser dargestellt werden können.

Künstliche Neuronale Netze (KNN) sind gewichtete Graphen bestehend aus einem Verbund künstlicher Neuronen. Es ist ein abstraktes Modell der Informationsverarbeitung, dass durch biologische neuronale Netze motiviert ist. Ein *künstliches Neuron* (vgl. Abb. 3.12) gewichtet den Einfluss von Eingaben und bestimmt mit der *Aktivierungsfunktion* dessen Lernverhalten. Indem neue Neuronen oder deren Verbindungen hinzugefügt oder gelöscht, Gewichtungen verändert oder Schwellenwerte angepasst werden, kann das KNN trainiert werden. Die Struktur und Anordnung der Neuronen eines KNN kann zweckmäßig verschieden ausfallen. Eine typische Struktur ist nach [Wood, 2008, S. 2] das mehrschichtige Feedforward-Netz (*Feedforward Multi-Layer Perceptron*), in dem Signale ein dreischichtiges Netz gerichtet durchlaufen: *Input Layer*, *Hidden Layer* und *Output Layer*. Eine ausgeführte Geste wird in ihre Bestandteile zerlegt, die als Merkmale der Klassifikation dienen. Diese Daten werden an die Neuronen der Eingabeschicht (*Input Layer*) weitergegeben. In [Nygård, 2010] werden zwei Ansätze vorgestellt - zum einen die Klassifikation basierend auf Richtung und Winkeln, zum anderen die Eingabe von Bildpixeln. In der versteckten Schicht (*Hidden Layer*) wird auf Basis antrainierter Gewichtungen die eigentliche Klassifikation der Geste vorgenommen. Jedes Neuron der Ausgabeschicht (*Output Layer*) repräsentiert letztlich eine Klasse von Gesten. Das KNN ist ein leistungsfähiger Ansatz der Gestenklassifikation, der sich ähnlich wie HMM durch Training zu optimieren ist. [Wood, 2008]

Künstliche Neuronale Netze

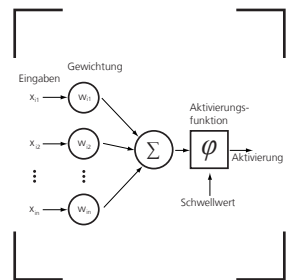


Abbildung 3.12:
Modell eines künstlichen Neurons

Während künstliche neuronale Netze Muster und ihre Verarbeitung erst erlernen müssen, baut *Fuzzy-Logik* auf vorhandenem Wissen auf. Im Gegensatz zur *Prädikatenlogik*, in der eine klare Abgrenzung von wahren und falschen Aussagen vorgenommen werden kann, führt Fuzzy-Logik unterschiedliche Zugehörigkeitsgrade zu einer Eigenschaft ein, um Aussagen zu approximieren. (vgl. [Wood, 2008]) Die Theorie wird in der Gestenerkennung dafür genutzt, absolute Messdaten in unscharfe Formalismen zu überführen. In [Bimber et al., 2000] wird eine Klassifikation von geometrischen und dynamischen Eigenschaften von Gesten, wie zum Beispiel die Länge eines Gestenpfads, die relative Ausrichtung oder Aussagen über die Geradlinigkeit einer Bewegung vorgenommen. Jede Eigenschaft ist über vergleichbare Wertemengen definiert, die eine graduelle Übereinstimmung einführen. HENZEN beschreibt einen ähnlichen Ansatz nach JORGE ET AL., der auf Basis dieser Theorie ebenso eine Merkmalsextraktion vornimmt und die passende Geste über einen Entscheidungsbaum ermittelt. (vgl. [Henzen, 2010])

Fuzzy-Logik

Dynamic Time Warping (DTW) ist ein *Template-Matching-Algorithmus*, der auf *dynamischer Programmierung* basiert und hauptsächlich in der Sprach-Mustererkennung eingesetzt wird. (vgl. [Wobbrock et al., 2007]) Er ist jedoch genauso gut auf Gesten anwendbar (vgl. [Scholliers et al., 2011]). Dabei stellen *Referenzmuster* (Templates) das Vokabular des Systems dar, welches erkannt werden kann. Der Vergleich der Eingabe (*Testmuster*) wird mit den Referenzmustern abgeglichen. Dabei stellt sich das Problem, dass Eingaben vorallem in ihrer zeitlichen Dimension variieren und ein direkter Abgleich meist zu negativen Ergebnissen führt. In der Sprache sind es vorallem Vokale, die oft unterschiedlich lang ausgesprochen werden und deshalb vom angenommenen Referenzmuster abweichen. Dies kann auf die Ausführung von Gesten übertragen werden. Der Algorithmus verarbeitet zeitabhängige Eigenschaften, wie zum Beispiel einzelne Positionsdaten eines Gestenpfads und stellt sie den bekannten Referenzmustern gegenüber. Das Prinzip der dynamischen Programmierung optimiert die Vergleichslogik, indem zunächst Teilprobleme definiert werden, die dann zu einer Gesamtlösung kombiniert werden. Der Algorithmus erzielt damit eine gute Erkennungsrate. Nachteilig ist jedoch die Performanz bei der Erkennung und ein hoher Speicherplatzbedarf für die Menge an Referenzmustern, was insbesondere auf mobilen Endgeräten ein Problem darstellt. (vgl. [Wobbrock et al., 2007] und [Wood, 2008])

Dynamic Time Warping

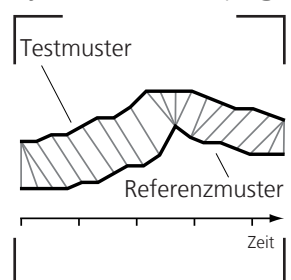


Abbildung 3.13:
Nicht-lineares Matching von Referenz- und Testmuster in DTW

Merkmalsextraktion und Gewichtung nach RUBINE

RUBINE wendet für die Erkennung von einfachen Maus- und Stiftgesten eine Methode an, die auf der Berechnung von sogenannten *Feature*-Vektoren basiert. [Rubine, 1991] Jede Geste wird durch einen solchen Vektor repräsentiert, der mehrere ihrer Eigenschaften (*Features*) beschreibt. 13 Features werden in [Rubine, 1991] insgesamt vorgeschlagen, um eine ausreichende Unterscheidung von Gesten vornehmen zu können, darunter beispielsweise die Dauer, Länge oder Geschwindigkeit in der sie gezeichnet wird. Das Verfahren stützt sich auf einer Menge von antrainierten Beispielen für die jeweilige zu klassifizierende Geste. Auf dieser Basis wird eine Gewichtung ermittelt, die in die sogenannte *Diskriminanzfunktion* einfließt. Über den größten errechneten Wert kann schließlich die zugehörige Klasse bestimmt werden.

Richtungs-basiertes Pattern Matching

Ein einfacher auf Ausführungsrichtungen basierender Ansatz wurde in Kapitel 3.3.1 im Verfahren von *Moousture* vorgestellt. Der Erkennungsalgorithmus nimmt nach der ausführlichen Beschreibung in [Nygård, 2010] eine Berechnung von Winkeln für jeden Punkt des Gestenpfads vor und ordnet ihm daraufhin einen der acht Richtungsvektoren zu. Dieses Vorgehen ist dem *SiGeR*-Algorithmus (Simple Gesture Recognizer) nach [Swigart, 2005] im Grunde sehr ähnlich. Basierend auf einer derartigen Formalisierung ist es möglich ein stringbasiertes Pattern-Matching durchzuführen, wie es auch in [Henzen, 2010] im Zusammenhang mit dem Ansatz von STIEFMEIER ET AL. beschrieben ist. Grundlage für die Erkennung von Ähnlichkeiten ist die Berechnung der *Levensthein*-Distanz. [Nygård, 2010]

\$1 Recognizer

WOBBROCK ET AL. schlagen mit dem *\$1 Recognizer* eine koordinatenbasierte, bildliche Auswertung von Einzelpfad-Gesten vor. Die Erkennungsmethode setzt lediglich eine Koordinatenliste als Referenzmuster voraus, auf deren Basis eine Ähnlichkeit zu den Punkten der Eingabe (*Kandidat*) festgestellt wird. Inbegriffen ist eine vereinheitlichende Vorverarbeitung, die eine einfachere Berechnung ermöglicht und Abweichungen bezüglich der Orientierung, Größe und Position der Geste toleriert.

"It supports configurable rotation, scale, and position invariance, does not require feature selection or training examples, is resilient to variations in input sampling, and supports high recognition rates, even after only one representative example." [Wobbrock et al., 2007]

Der Algorithmus durchläuft im Wesentlichen vier Schritte:

1. *Resampling*: Gesten werden durch ungleichmäßig verteilte Punkte repräsentiert, was vorallem auf unterschiedliche Ausführungsgeschwindigkeiten zurückzuführen ist. Um die Vergleichbarkeit zu gewährleisten, werden die Punkte auf dem gegebenen Pfad des Kandidaten neu verteilt. Alle Vorlagen und Kandidaten werden damit durch die gleiche Anzahl von Punkten repräsentiert.
2. *Vereinheitlichung der Orientierung*: Der Rotationswinkel des Kandidaten wird über das Zentrum der Geste und dessen ersten Punkt ermittelt. Der Kandidat wird anschließend auf 0° um sein Zentrum gedreht.
3. *Skalieren und Verschiebung*: Die Dimension und Position des Kandidaten wird anhand eines festgelegten Referenzquadrates und Referenzpunktes vereinheitlicht.
4. *Template-Matching*: Der Kandidat wird mit allen Vorlagen verglichen, indem die durchschnittliche Abweichung ihrer zugehörigen Punkte errechnet wird. Die minimale Pfad-Distanz wird dann normiert und schließlich bewertet.

Im Rahmen eines *Google Research* Projektes stellt LI eine Erweiterung des Algorithmus vor, das unter dem Namen *Protractor* veröffentlicht wurde. [Li, 2010b] Die vorgeschlagenen Verbesserungen betreffen insbesondere die Möglichkeit den Algorithmus so zu konfigurieren, dass verschieden ausgerichtete Gesten unterschieden werden. Diese als *orientation sensitive* benannte Eigenschaft wird gegeben, indem die Geste nicht mehr einheitlich auf 0° gedreht wird, sondern zwischen acht Hauptausrichtungen unterschieden wird. Die naheliegendsten Hauptrichtung bestimmt die Orientierung des Kandidaten. Weiterhin werden normierte Vektoren für die Repräsentation der Gesten verwendet. Basierend auf der Berechnung minimaler Winkel zwischen den Vektoren, wird die Ähnlichkeit zwischen der Geste und den Vorlagen festgestellt. Mit dieser Methode übertrifft Protractor den \$1-Algorithmus mit einer höheren Erkennungsgeschwindigkeit, einer besseren Erkennungsrate und einen um 1/4 geringeren Speicherverbrauch. [Li, 2010b, S. 3]

*Verbesserungen des
\$1Recognizer nach LI*

Die vorgestellten Ansätze konzentrieren sich vorrangig auf die Erkennung von Einzelpfad-Gesten. Eine Vielzahl symbolischer Gesten lässt sich allerdings nur über mehrere Ausführungspfade beschreiben, die im Verbund zu interpretieren sind. Neben der Erkennung ein- und mehrpfadige Gesten als wesentliches Merkmal des Ansatzes im \$N-Erkennungsverfahren [Anthony und Wobbrock, 2010], wurden Verbesserungen hinsichtlich der Erkennung eindimensionaler Gesten, der Orientierung einer Geste und der Verarbeitungsgeschwindigkeit vorgenommen.

*Zusammengesetzte
Gesten mit \$N*

Neben den genannten Verfahren werden in [Henzen, 2010] weitere Ansätze beschrieben, wie zum Beispiel das auf einer Rasterung des Eingabebereichs basierende Matching nach CELENTANO ET AL., logikbasierte Funktionsvektoren nach CHO (im Ansatz ähnlich wie die Methode nach [Rubine, 1991]), der Vergleich XML-basierter Beschreibungen nach SOLMS oder die Analyse von Ecken und Krümmungen eines Gestenpfads nach SEZGIN.

Weitere Verfahren

3.3.3.1 Zusammenfassung und Vergleich der Methoden

Aus der Beschreibung der Methoden gehen drei wesentliche Aspekte der Gestenerkennung hervor, die in Hinblick auf den Entwurf eines sprachbasierten Web-Frameworks zu bewerten sind:

Bewertungskriterien

- *Repräsentatives Datenmodell*
Es ist ein geeignetes Datenmodell zu finden, das aus der formalen Gestenbeschreibung abgeleitet werden kann. Insbesondere in mobilen Webanwendungen spielt dabei der Speicherverbrauch eine wichtige Rolle.
- *Merkmalsextraktion und -selektion*
Die elementaren Ereignisse einer gestischen Eingabe sind in Hinblick auf das Datenmodell zu verarbeiten und in eine vergleichbare Repräsentation zu überführen. Es müssen entsprechend relevante Merkmale aus einer Ereignissequenz extrahiert werden. Da dies zur Eingabezeit geschieht, ist der Aufwand, der sich damit ergibt, und die daraus resultierende Verarbeitungsgeschwindigkeit ein wichtiger Faktor bei der Wahl einer Erkennungsmethode.
- *Klassifikationsverfahren*
Das Verfahren der Zuordnung einer Eingabe zu einer definierte Geste basierend auf einer Vergleichslogik ist das Bindeglied zwischen Datenmodell und Merkmalsextraktion. Bewertungsgrundlage sind dabei die Erkennungsrate und Performanz.

Klassifikation der Verfahren

Prinzipiell können *statistische Klassifikationsverfahren* (HMM, KNN, Rubine-Algorithmus), die auf einem trainierten Modell aufbauen, und *Template-Matching-Verfahren* unterschieden werden (DTW, SiGer, \$1Recognizer und Erweiterungen), die eine koordinatenbasierte, strukturelle oder syntaktische Ähnlichkeitsanalyse auf Basis vergleichbarer Vorlagen vornehmen. Tabelle 3.1 gibt einen zusammenfassenden Überblick über die beschriebenen Verfahren.

Ansatz	Datenmodell	Merkmalsextraktion	Klassifikation
HMM	Modellierung von Zuständen, Training der Übergangswahrscheinlichkeiten	modellabhängig	Stochastische Berechnung
KNN	Modellierung eines Neuronennetzes, Training der funktionalen Parameter	modellabhängig	Durchlaufen von Filter- bzw. Aktivierungsfunktionen
Fuzzy-Logik	Modellierung von Wertebereichen mit unscharfen Klassifikationsgrenzen für verschiedene Ausführungseigenschaften	Merkmalspezifische Berechnungen (nach [Bimber et al., 2000] etwa 27)	Regelsystem, z.B. Entscheidungstabelle oder -baum
Rubine	Repräsentation der Gestenmerkmale als Feature-Vektor, mehrere Referenzmuster pro Geste als Datenbasis erforderlich	nach [Rubine, 1991] sind 13 Berechnungen für die Merkmalsextraktion vorzunehmen	Diskriminanzanalyse basierend auf Feature-Vektoren und Gewichtung der Gestenklassen
DTW	Zeitabhängige Signale als Referenzmuster, hoher Speicherbedarf	Filterung der Eingabedaten nach relevanten Merkmalen, bspw. Positionsdaten	Template-Matching basierend auf dynamischer Programmierung
SiGer	Formale Repräsentation der acht möglichen Ausführungsrichtungen als Zeichenkette	Extraktion der Ausführungsrichtungen (beispielsweise über Winkelberechnung)	Ermittlung der Abweichung über Levenshtein-Distanz
\$1/ \$N/ Protractor	Liste von Koordinaten	Filterung und Normierung des Ausführungspfades	Berechnung der durchschnittlichen Abweichung der Koordinaten

Tabelle 3.1: Methoden der Gestenerkennung im Überblick

Statistische Verfahren

Statistische Verfahren wie HMM und KNN müssen vorab trainiert werden, was sehr langwierig und aufwändig sein kann. Sie sind sehr mächtig, erfordern aber ein besonderes Verständnis über die zugrunde liegende Theorie. Die Modelle werden anwendspezifisch konstruiert, sind aber dadurch nicht besonders flexibel. Dennoch zeichnen sie sich durch ihre besonders gute Erkennungsrate und Verarbeitungsgeschwindigkeit aus. (vgl. [Wood, 2008]) Nichtsdestotrotz werden sie in [Wobbrock et al., 2007, S. 2] als unpraktisch für prototypische Implementierungen erachtet, vor allem wegen des unumgänglichen Trainingsvorgangs. Dieser Aspekt ist auch im Zusammenhang mit der Generierung des Datenmodells aus einer formalen Spezifikation heraus problematisch zu sehen. Des Weiteren ist die Konzeption und Umsetzung der Algorithmen komplex, was die Programmierung und die Fehlerdiagnose erschweren. [Wobbrock et al., 2007, S. 2] Das Arbeiten mit Vektoren nach [Rubine, 1991] ist hingegen leichter umzusetzen, erfordert aber ebenso mehrere Musterbeispiele, die zuvor in das System eingelesen werden müssen. Fuzzy-Logik ist dem Ansatz von KNN sehr ähnlich, kommt aber ohne Training aus. Herausfordernd ist

jedoch die Implementierung eines komplexen Regelsystems, das möglichst viele Unterscheidungsmerkmale zur Klassifikation einbezieht.

Template-Matching-Verfahren basieren auf einer festgelegten Datenbasis. Jede Geste wird durch ein Referenzmuster repräsentiert, das mit der Eingabe verglichen werden kann. Nachteil dessen ist, dass eine solche Datenbank mit zunehmender Anzahl von Gesten sehr speicherintensiv ist. Dies stellt insbesondere in DTW ein Problem dar. Die erforderliche Erkennungszeit mit DTW ist dafür aber im Vergleich zu HMM und KNN geringer (vgl. [Wood, 2008]). Dennoch bewerten WOBBEROCK ET AL. den Algorithmus wie folgt: *“Dynamic programming methods are computationally expensive and sometimes too flexible in matching [...], and although improvements in speed are possible [...], these improvements put the algorithms well beyond the reach of most UI designers and prototypers.”* [Wobbrock et al., 2007, S. 2] In dieser Hinsicht sind Algorithmen wie SiGer eindeutig besser geeignet, denn die Definition von Gesten erfolgt auf Basis verständlicher Ausdrucksformen, der Speicherbedarf ist dadurch niedrig und der Vergleichsalgorithmus effizient. Die Performanz dieses Ansatzes richtet sich hier maßgeblich nach der Methode, die die Ausführungsrichtung ermittelt. WOBBEROCK, LI und ANTHONY schlagen statt der Übersetzung in formale Parameter eine Normierung der Koordinaten vor und schaffen damit eine tolerante Vergleichsbasis auf elementarer Ebene. ([Wobbrock et al., 2007], [Li, 2010b], [Anthony und Wobbrock, 2010]) Einer hohen Erkennungsrate, die mit DTW vergleichbar ist [Wobbrock et al., 2007] steht die Fähigkeit gegenüber, hohe Abweichungen von den Vorlagen als Fehleingabe zu interpretieren. Dies leistet der Algorithmus nicht. Eine differenzierte Klassifikation der Gesten hängt in diesem Ansatz maßgeblich von den Vorlagen ab. Die Normierung führt dazu, dass bestimmte Unterscheidungsmerkmale wie zum Beispiel die Größenverhältnisse nicht in die Erkennung einfließen können. Hinsichtlich der Erkennung von Multitouch-Gesten fehlt den richtungs- und koordinatenbasierten Template-Matching-Methoden grundsätzlich der zeitliche Bezug. Es erfolgt demnach lediglich eine Auswertung der Bewegungsform. Dennoch eignen sich diese Verfahren am ehesten für den Einsatz in einem Web-Framework, vorallem in Hinblick auf die Anwendung in mobilen Webanwendungen und hinsichtlich ihrer Flexibilität und Erweiterbarkeit.

Template-Matching-Verfahren

3.4 APPLIKATION

In diesem Kapitel werden Aspekte des *Interaktionsdesigns* auf Anwendungsebene in Betracht gezogen. Das Interaktionsdesign konzentriert sich nach der Definition in [Cooper et al., 2007] auf die Konzeption des Verhaltens der Applikation, sowie alle Elemente, die dieses Verhalten kommunizieren. Nach [Dorau, 2011] bedeutet Interaktionsdesign nicht nur die Entwicklung eines Bedienkonzepts, sondern es müssen ebenso ästhetische Aspekte und das mentale Modell einbezogen werden. Dies liegt im Verantwortungsbereich der Web-Entwickler. Das Web-Framework soll diese Aufgabe unterstützen, ohne die gestalterischen Freiheiten einzuschränken. Ziel dieses Kapitels ist es demnach nicht Vorgaben für ein gutes Design zu erarbeiten oder *Best-Practices* wiederzugeben, sondern ein grundlegendes Verständnis für die Anwendung von Gesten auf dieser Konzeptionsebene zu entwickeln. Das ist notwendig um den Entwurf des Frameworks zweckmäßig und benutzerorientiert zu überdenken. Im Folgenden werden dafür Interaktionstechniken verglichen (Abschnitt 3.4.1), sowie Eigenschaften und Besonderheiten von Touch-Interfaces näher beleuchtet (Abschnitt 3.4.2). Des Weiteren beschäftigt sich dieses Kapitel mit der Beziehung zwischen elementarer Eingabe und zu lösender Aufgabe (Abschnitt 3.4.3) und präsentiert konkrete Beispiele für die Anwendung von Gesten (Abschnitt 3.4.4).

3.4.1 Vergleich von Maus-, Stift- und Touch-Interaktion

*Bezugssystem: Relative
und absolute
Positionierung*

Der offensichtlichste Unterschied zwischen der Bedienung mit der Maus und der direkten Eingabe mit dem Finger liegt in deren Bezugssystem. Die traditionelle Maus ist ein Werkzeug, das relativen Bewegungen der Hand auf die Darstellungsfläche des Interface projiziert. Ein Cursor hilft dabei die aktuelle Position und Bewegung nachzuvollziehen. Die Führung der Maus erfolgt intuitiv auf Basis der visuellen Rückmeldung, erfordert nach [Geißler, 2001, S. 32] jedoch eine höhere Konzentration, um Hand, Maus, Bildschirm, Auge und Gehirn zu koordinieren. Touchscreens bauen auf einem absoluten Positionierungssystem auf, in dem eine direkte Beziehung zwischen Eingabeort und fokussiertem Bereich der Benutzeroberfläche besteht. Die Koordinierungsarbeit des Anwenders wird dadurch gestützt, dass sich Eingabebereich und Darstellungsfläche decken. Im Gegensatz zu Mauszeigersystemen, ergeben sich daraus konkrete, klar definierte Systemgrenzen.

*Zustände absoluter und
relativer Eingaben*

WIGDOR ET AL. nehmen ein Vergleich anhand von Zustandsmodellen vor (vgl. [Wigdor und Wixon, 2011, S. 65 ff.]). Touch-Eingaben beziehen sich lediglich auf zwei Zustände (*Idle* und *Active*). Eine Berührung ist eine messbare Größe und versteht sich daher als Aktivierung des Systems. Ein Kontakt ist da oder eben nicht. Die Mauszeigersteuerung kann durch drei Zustände beschrieben werden (*Out of Range*, *Tracking* und *Engaged*) und ist bereits ohne zutun des Users aktiv, denn die Präsenz des Cursors ist unter dem verbreiteten GUI-Grundprinzip WIMP (*Windows-Icons-Menus-Pointer*) vorgegeben. Ohne diese Orientierungshilfe wäre eine genaue Positionierung auf Basis relativer Bewegungen sehr schwer, man könnte behaupten unmöglich. Das Tracking übersetzt die Bewegungen auf die Cursorposition, leitet jedoch keine bedeutenden Systemaktionen ein. Hierfür ist ein weiterer ausdrucksstärkerer Zustand des Eingabesystems vorgesehen, der durch das Betätigen einer Maustaste oder vergleichbare Schaltmechanismen initiiert wird. Ein derartiger Zustand ist essentiell, um eine gestische Handlung sicher abzugrenzen und zu erkennen. Am Beispiel des Trackpads ist zu sehen, dass ein berührungssensitives Ein-

gabegerät durchaus fähig ist ein Zeigesystem wie die Maus zu emulieren (vgl. [Wigdor und Wixon, 2011, S. 69]). Die Anwendung des 2-Zustände-Modells der Touch-Interaktion für die absolute Positionierung des Cursors ist dahingegen schwieriger, wenn Ein- und Ausgabebereich separiert sind. Der Nutzer müsste die Benutzeroberfläche vor seinem geistigen Auge auf die Eingabefläche projizieren und die Position und Größenverhältnisse schätzen, was hinsichtlich der Genauigkeit dieser Methode problematisch ist.

Stiftbasierte Eingabesysteme sind weitaus flexibler einsetzbar. Einige Grafiktablets erlauben es beispielsweise zwischen relativer Zeigersteuerung und absoluter Eingabe (im Rahmen von Tablet-Grenzen) umzuschalten. Dies setzt entsprechende Technologien und Hardware voraus, mit denen sich die genannten Zustände emulieren lassen. Beispielsweise werden Schaltflächen an den Stift angebracht, um eine Klickaktion zu ermöglichen. Eine bestimmte Druckstärke kann ebenso einen solchen Zustandsübergang einleiten. Diese Arbeit konzentriert sich hauptsächlich auf die Verwendung von Stiften als Ersatz des Fingers in direkten Touch-Eingabesystemen, mit dem Ziel, sich die Präzision dieses Hilfsmittels zu nutze zu machen. Der Fokus liegt damit allein auf den Positionsdaten, die das Gerät bei direkter Berührung liefert. Stifte sind ein gebräuchlicher Alltagsgegenstand für schriftliche und bildnerische Aufzeichnungen, weshalb sie vorallem für symbolische Gesten geeignet erscheinen.

Stifteingabe

Bezüglich der Präzision in direkt zu bedienenden Benutzeroberflächen ist der Stift gegenüber der Interaktion mit dem Finger zu bevorzugen. Ein Mauscursor kann dem gegenüber jedoch noch genauer positioniert werden, da eine pixelgenaue Virtualisierung des Zeigeelements vorliegt. (vgl. [Geißler, 2001, S. 32]) Ein entscheidender Unterschied ist auch die Möglichkeit Korrekturen bei der Fokussierung von Objekten vorzunehmen, basierend auf der Trennung von Zuständen für die Positionierung und Ausführung. Korrekturen sind jedoch auch immer mit einem höheren Aufwand und Zeit verbunden, auch wenn die Übertragung der Mausbewegungen grundlegend schneller ist als bei anderen Eingabegeräten (vgl. [Forlines et al., 2007] und [Wigdor und Wixon, 2011, S. 118]). In dieser Hinsicht ist das Selektieren von Elementen mit direkter Berührung sicherer, da der Koordinierungsaufwand geringer ist. Dies gilt aber nur solange die Angriffsfläche des Elements groß genug ist und in unmittelbarer Nähe ist. (vgl. [Forlines et al., 2007]) Die Fehlerrate erhöht sich jedoch mit zunehmender Distanz. In [Henzen, 2010, S. 62 ff.] werden diesbezüglich vier Interaktionszonen unterschieden, die sich an markanten Erreichbarkeitsgrenzen orientieren und dementsprechend zu adaptieren sind. Sobald Armbewegungen einbezogen werden müssen, um entfernte Bereiche des Touch-Interface zu bedienen, erhöht sich die Zeit für die Bewältigung von Interaktionsaufgaben (vgl. [Forlines et al., 2007]).

Präzision und Erreichbarkeit

Die Maus-Interaktion erscheint für primitive Aufgaben, wie das Bestätigen einer Aktion mit einem Klick oder Öffnen von Ordnern mit einem Doppelklick, die effizientere Alternative zu sein. Nach [Forlines et al., 2007] bevorzugten 9 von 12 Testpersonen diese Eingabetechnik für einfache Aufgaben. Dies ist damit zu begründen, dass die Maus mehr Komfort bietet und sich bereits kleinste Bewegungen des Handgelenks übersetzten lassen. Die Ausführung pfadbasierter Gesten ist hingegen komplizierter: *“Open up a drawing program and try drawing a perfect (heck, even passable) circle. Now grab a pen and a piece of paper. Which one was faster, and which one looks better?”* [Wigdor und Wixon, 2011, S. 118]. Das Interaktionskonzept der Maus ist nicht dafür ausgelegt, um kontrolliert Pfade zu skizzieren, ebenso wie es dem natürlichen Wesen von Stiften und Berühreingaben widerspricht, den *Tracking*-Zustand einer Maus zu emulieren.

Pfadbasierte Gesten

3.4.2 Besonderheiten des Multitouch-Interface

Wie zu Beginn dieser Arbeit bereits erklärt wurde, ist die Interaktionskonzepte von Web-Anwendungen historisch bedingt auf Eingabegeräten abgestimmt, die das Zustandsmodell einer klassischen Maus abbilden. Für die Adaption auf berührungsbasierte Eingabeschnittstellen sind Besonderheiten des Multitouch-Interface zu betrachten.

Fat-Finger-Problem

Das Fat-Finger-Problem bezeichnet die Herausforderung des Interaktionssystems die größere Kontaktfläche des Fingers in eine präzisere Zeigeposition zu übersetzen. Nach [Dorau, 2011, S. 189] wird dazu meist der geometrische Mittelpunkt der Kontaktfläche berechnet. Es existieren jedoch unterschiedliche Verfahren für das Tracken eines Kontaktes. (vgl. [Holz und Baudisch, 2011]) Grafisch lässt sich das Problem mit der vergrößerten Darstellung interaktiver Elemente lösen. Diesem Ansatz steht jedoch der Trend zu kleineren Touchscreens in mobilen Geräten gegenüber. (vgl. [Wigdor und Wixon, 2011, S. 74 ff.]) Nach [Dorau, 2011, S. 191 f.] kann mit einem nicht sichtbaren Toleranzbereich (*hit area*), der das interaktive Element umgibt erreicht werden, dass die Bedienung auch bei leichtem Verfehlen funktioniert. Der Erkennungsbereich einer Geste ist demnach größer als das dargestellte Element. Darüber hinaus ist ein größerer Abstand zwischen interaktiven Elementen wie Hyperlinks zu empfehlen, um Fehleingaben und Verwechslungen entgegenzuwirken.

Verdeckung

Ein weiterer beachtenswerter Punkt bei der Gestaltung von Multitouch-Anwendungen ist die Verdeckung von Bildschirmbereichen. Das betrifft zum einen die Verdeckung, die mit dem Fat-Finger-Problem einhergeht, zum anderen aber auch die Teile die bei der Interaktion vom Rest der Hand überschattet werden. Es ist demnach darauf zu achten, dass die visuelle Rückmeldung und wichtige Informationen, die für weitere Aktionen entscheidend sind auch während einer Eingabe im sichtbaren Bereich liegen.

Gorilla-Arm-Problem

Eine Folge der absoluten Positionierung der Finger auf dem Touchscreen ist die schnelle Ermüdung, hervorgerufen durch häufige Bewegungen, Auf- und Absetzen von Hand und Arm. Das unter der Bezeichnung Gorilla Arm bekannte Phänomen ist vorallem bei vertikal aufgestellten und großen Displays ein Problem. [Teiche et al., 2009] Aber auch kleinere Bewegungen mit einem Finger können die Muskeln auf Dauer überstrapazieren. Das Ablegen der Hand oder Aufstützen des Arms kann hier Abhilfe schaffen, allerdings ist hierbei zu befürchten, dass Inhalte verdeckt werden oder Aktionen ungewollt ausgelöst werden.

3.4.3 Aktionen und Anweisungen

Differenzierung von Anweisung und Aktion

Eine Geste ist, wie schon in Kapitel 2.2.2 erfasst wurde, eine Handlung, die einer kommunikativen Absicht des Anwenders folgt. Aufgabe des Systems ist, sie als konkrete *Anweisung* zu interpretieren und als Konsequenz eine *Aktion* zu generieren, die möglichst den Erwartungen des Akteurs entspricht. Wichtigster Aspekt einer Aktion ist das Feedback, das dem Anwender Sicherheit im Umgang mit der Anwendung gibt. Gesten können demnach auch als Vermittler zwischen Anweisung und Aktion verstanden werden.

Dimension der gestenbasierten Interaktion

Diesbezüglich sind Gesten in einem Modell von Abstraktionsgraden zwischen System und Anwender einzuordnen. Abbildung 3.14 fasst die konzeptionellen Ansätze nach [Kammer et al., 2010], [Cooper et al., 2007] und [W3C, 2010b] zusammen.

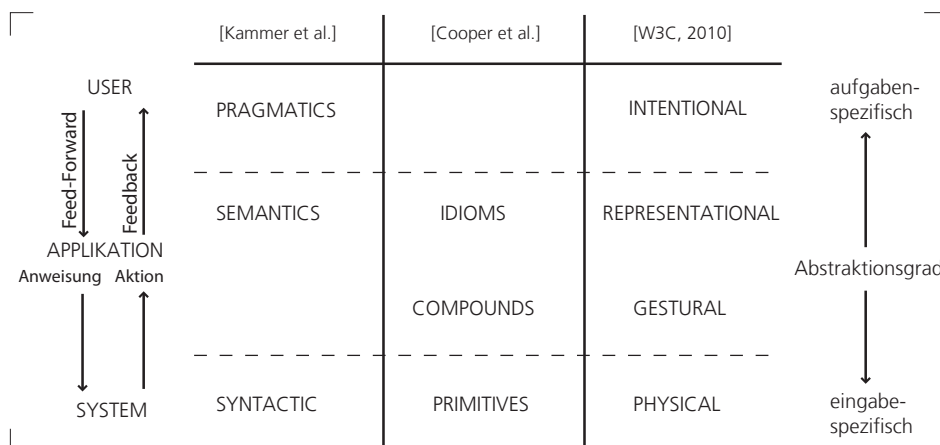


Abbildung 3.14: Gegenüberstellung der Abstraktionsgrade nach [Kammer et al., 2010], [Cooper et al., 2007] und [W3C, 2010b]

Das Modell nach [Kammer et al., 2010] unterscheidet drei Aspekte im Fokus der Semiotik als wissenschaftlichen Rahmen für die Formalisierung von Gesten. *Syntactics* beschreiben Gesten in ihren elementarsten erfassbaren Einheiten und ihrer Kompositionsvielfalt im Bereich der Möglichkeiten des Eingabegerätes. Im Kontext der Systemanwendung wird den Gesten eine Bedeutung zugeordnet (*Semantics*), mit der ein Bezug zur Aufgabe geschaffen wird. Diesbezüglich wurden Klassifikationsansätze in Kapitel 2.2.3 vorgestellt. *Pragmatics* umfassen das mentale Modell des Anwenders, insbesondere dessen Absichten und Erwartungen im Dialog mit dem System.

Modell nach KAMMER ET AL.

COOPER ET AL. betrachten die Dimensionen der Interaktion in Zusammenhang mit Bedienelementen der Benutzungsschnittstelle und den Möglichkeiten der darauf anwendbaren Aktionen. Zu den *Primitives* gehören elementare Eingaben wie das Betätigen einer Taste auf dem Eingabegeräte oder eine einfache Zeigehandlung. Die Konsequenz ist ein elementares Feedback, wie zum Beispiel die Neupositionierung des Cursors. *Compounds* sind Kompositionen elementarer Anweisungen, wie zum Beispiel die Selektion verschiedener Bedienelemente oder ein Doppelklick, um ein Programm auszuführen. Unter *Idioms* sind komplexe aufgabenspezifische Anweisungen und Aktionen zusammengefasst, wie das Zeichnen, Erzeugen oder Löschen von Objekten: "[...] information related to the user's work patterns and goals, and not specifically to the computerized solution." [Cooper et al., 2007, S. 281]

Modell nach COOPER ET AL.

Nach der Web Events Working Group Charter unter [W3C, 2010b] wird eine ähnliche Differenzierung von Gesten vorgenommen. Hinsichtlich der Umsetzung der Charter wird deutlich, dass es sich bei diesem Modell konkret um ein eventbasiertes Schichtenmodell handelt. Auf Systemebene (*physical*) werden elementare Eingaben, wie ein oder mehrere Kontakte auf der sensitiven Oberfläche, verarbeitet. Weiterhin sind gerätespezifische Eingabeparameter zu berücksichtigen, wie die Größe und Form des Kontaktes oder variierender Druck. Auf nächsthöherer Ebene (*gestural*) werden komplexe Gesten, die sich aus der Kompositionen elementarer Ereignisse ergeben, zusammengefasst. Erst im *representational*-Layer erfolgt eine semantische Anreicherung, die der Aufgabe gerecht wird. Die Trennung zwischen *gestural* und *representational* ist vorallem deshalb wichtig, weil eine 1-zu-N-Beziehung zwischen den Schichten vorliegt, die sich vorallem durch Variationen der Ausführung ergibt (vgl. [Hinrichs und Carpendale, 2011]). Beispielsweise ist die repräsentative Geste, die bezüglich der formalen Umschreibung ZOOM die Intention des Anwenders und die resultierende Aktion einbezieht, durch eine PINCH-Geste, ein DOUBLE-TAP oder

Abstraktionsgrade des W3C

eine andere anwendungsspezifische Geste auszuführen. Außerhalb des technischen Systems sind die Assoziationen zum Bedienelement entscheidend für die Wahl einer Geste (*intentional*), genauso wie dessen Reaktionsverhalten. HINRICHS ET AL. beobachten im Rahmen einer Studie, dass die Wahl nacheinander ausgeführter Gesten oftmals davon abhängt welche Aktionen zuvor ausgeführt wurden. Aktionen gehen dabei oft fließend ineinander über und bilden einen Verbund mehrerer Gesten. Beispielsweise werden auf einem Multitouch-Tisch manipulierbare Objekte wie Bilder für eine nähere Betrachtung in den Einflussbereich des Anwenders herangezogen und dabei bereits gedreht und vergrößert. (vgl. [Hinrichs und Carpendale, 2011])

3.4.4 Anwendung von Gesten

Grundlage für die Entwicklung neuer Bedienkonzepte ist das Wissen um gebrauchstaugliche Gesten. Bezeichnend sind ihre Ausdrucksstärke und ein verbreitetes Verständnis um ihrer Bedeutung. Ihr Durchsetzungsvermögen gründet auf einer starken metaphorischen Beziehung und Selbstbeschreibungsfähigkeit, aber auch auf gezielter Vermarktung. Die Umsetzbarkeit dieser Gesten mit dem Framework ist aus diesem Grund von besonderer Bedeutung. Angelehnt an die aufgabenbezogene Darlegung von Anwendungskonzepten in [Dorau, 2011, S. 136], werden im Folgenden geeignete Gesten vorgestellt. In einigen Fällen lassen sich konkrete Web-Implementierungen finden, die im Kontext dieser Arbeit von besonderem Interesse sind.

Selektieren und Markieren

Die einfachste Form der Selektion ist eine Zeigehandlung, wie das Tippen (TAP) oder Halten des Fingers (HOLD) auf einem für derartige Anweisungen ausgezeichneten Objektes, wie zum Beispiel eine Checkbox. Oftmals geht diese Art der Selektion auch mit der Ausführung einer Funktion und Weiterleitung zu einer neuen Ansicht einher, wie es bei Links und Buttons der Fall ist. Für stiftbasierte Systeme eignet sich auch das Unterstreichen oder Umkreisen eines Objektes oder einer Textauswahl. [Geißler, 2001, S. 75] WIGDOR ET AL. weisen auf das Potenzial des Durchstreichens von Auswahloptionen (siehe Abb. 3.15) zur Adaption des traditionellen Selektionsprinzips für Stifteingaben hin [Wigdor und Wixon, 2011, S. 121 f.]. Für verteilte Objektmengen bietet sich eine umschließende Kreisbewegung an, die in diesem Sinne auch als LASSO-Geste bezeichnet wird (vgl. Abb. 3.8). Ein Beispiel dafür, dass primitive Anweisungen wie die Selektion in Verbindung mit weiteren Aktionen zu komplexen Gesten mit neuer Absicht konstruiert werden können, ist das Prinzip von DRAG&DROP (*Selektieren-Verschieben-Loslassen*).

Scrollen und Blättern

Das Scrollen von Ansichten gehört zum Grundkonzept von Fenstersystemen. Multitouch-Anwendungen adaptieren diese Form der Navigation durch direkte Manipulation einer Ansicht, beruhend auf dem Prinzip der Verankerung. [Dorau, 2011, S. 110 ff.] Die Assoziation, die damit verknüpft ist, ist das Verschieben eines Blattes Papier hinter den Fenstergrenzen. DORAU bezeichnet diese Geste auch als DRAG. Auf dieser Bewegungsform baut auch die FLICK-Geste auf, die jedoch schneller und kürzer ausgeführt wird und diesen Schwung auf die Scrollgeschwindigkeit überträgt. [Dorau, 2011, S. 138 ff.] Für die indirekte Eingabe von Gesten, beispielsweise mit dem Trackpad oder einer Multitouch-Maus, haben sich für das Scrollen Mehrfinger-Gesten durchgesetzt, die sich von der Positionierung des Cursors mit einem Finger abheben (vgl. [Apple, 2010] und [Apple, 2009b]). DORAU verweist in diesem Zusammenhang auf das Potenzial hin, die Anzahl der Finger für die Variation der Scrollgeschwindigkeit zu nutzen. Mehr Finger betonen in dem Fall die Absicht einer Geste

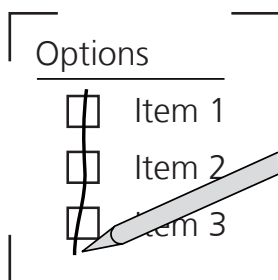


Abbildung 3.15:
Stiftbasierte
Mehrfach-Selektion
nach [Wigdor und
Wixon, 2011]

und sind ein Multiplikator für die Geschwindigkeit. [Dorau, 2011, S. 139] Angelehnt an das sogenannte *Clickwheel*, dass mit Apples iPod-Geräten bekannt geworden ist, stellt eine kreisförmige Bewegung (SPIN) zum Durchsuchen von Listen ebenso eine Option dar ([Dorau, 2011, S. 139] und [Saffer, 2008]). Das Blättern (SWIPE) von Ansichten ist eine diskrete Variante des Scrollens und eignet sich besonders für Anwendungsfälle, in denen der Nutzer zu festgelegten Navigationspunkten geführt werden soll. Beispielsweise beim Durchsuchen einer Bildergalerie oder die Änderung ganzer Ansichten. HUNTER ET AL. implementieren diese Geste als Grundnavigation ihrer Webanwendung. [Hunter et al., 2010] Statt die Seiten vertikal zu scrollen, werden sie ähnlich wie in einem Buch geblättert. Abbildung 3.16 veranschaulicht dieses Verhalten.

Skalieren und Zoomen

Der Bedeutungsunterschied zwischen Skalieren und Zoomen liegt im Ausführungskontext der Geste. Dies kann ein Objekt oder die allumfassende Ansicht der Anwendung sein. Die Ausführung selbst ist jedoch die gleiche - das Zusammenziehen oder Auseinanderspreizen der Finger (auch als PINCH und SHRINK bezeichnet). Mobile Webbrowser setzen die Geste für das Zoomen der gesamten Web-Seite um, so dass auch auf kleineren Bildschirmen sowohl ein Überblick als auch Details der Ansicht erfasst werden können. Alternativ führt auch das zweimalige Antippen der Ansicht (DOUBLE-TAP) zu einer diskreten Zoom-Stufe. (vgl. [Apple, 2011a]) Eine prototypische Umsetzung der Manipulation von Web-Elementen, wie zum Beispiel Bilder oder Videos, ist beispielsweise in [Robinson, 2010] und [Cailliau et al., 2009] zu sehen. (vgl. Abb. 3.17)

Drehen

Wie im Beispiel des Skalierens und Zoomens, ist auch das Drehen eines Objektes und das Rotieren zum Zweck eines Perspektivwechsels zu unterscheiden. In einer zweidimensionalen Darstellung hat sich als Multitouch-Geste eine rotierende Bewegung zweier Kontakte um ihren Mittelpunkt durchgesetzt, wie es dem Umgang mit physischen Objekten entspricht. [Dorau, 2011, S. 149] Anwendung findet die Geste zum Beispiel in Kartenansicht oder auf Medien-Objekten (vgl. Abb. 3.17), die zur besseren Ansicht ausgerichtet werden müssen. Unabhängig davon zeigt Abbildung 3.18 ein Beispiel, in dem ein interaktives Kreisdiagramm basierend auf JavaScript implementiert wurde. Darin muss das Diagramm zunächst rotiert werden, um Informationen zu den einzelnen Segmenten in Form eines darüber angeordneten Tooltips zu erhalten. Eine spezielle Herausforderung ist die perspektivische und objektorientierte Rotation in dreidimensionalen Ansichten. HAYES zeigt am Beispiel eines 3D-Würfels, implementiert auf Basis von Web-Standards, wie die Rotation für Tastatur-, Maus-, Stift- und Touch-Interaktion gleichermaßen realisiert werden kann. [Hayes, 2010] In Bezug auf einen Perspektivwechsel dominiert in Web-Browsern mobiler Endgeräte die Drehung auf Basis der Bewegungssensoren. Je nachdem, wie das Gerät ausgerichtet wird (*quer/ hochkant*), passt sich die Ansicht dem Format an (*portrait/ landscape*).

Öffnen und Schließen

Der Doppelklick hat sich für das Öffnen eines Ordners oder Aktivieren eines Programms in Desktop-Betriebssystemen bewährt und damit besonders eingepreßt. [Dorau, 2011, S. 152] In Multitouch-Systemen wird die Technik in Form des doppelten Antippens (DOUBLE-TAP) adaptiert. Oftmals genügt jedoch ein einfaches Antippen (TAP) des jeweiligen Objektes. Ein erweitertes Konzept zum Öffnen und Schließen existiert durch Anwendung der PINCH- und SHRINK-Geste, wie sie auch zum Skalieren und Zoomen verwendet werden. Die Intention des Anwenders mag dabei sein, sich einen Ordner näher anzusehen. Aufbauend auf der Stapel-Metapher wird diese Geste auf kategorisierte Bilder im iPad-Programm *Fotos* angewendet. In [Murray, 2010] wird gezeigt, dass dieses Konzept auch auf Basis von Web-Standards

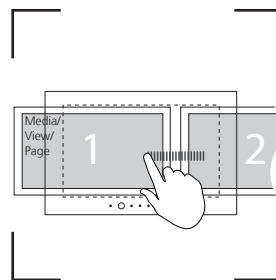


Abbildung 3.16:
SWIPE-Geste zum
Blättern in Ansichten



Abbildung 3.17:
Manipulation eines
Video-Objektes [Cailliau
et al., 2009]

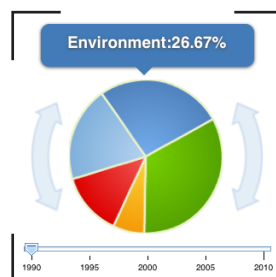


Abbildung 3.18:
Drehbares
Kreisdiagramm mit
Dojo mobile [Dojo,
2011]

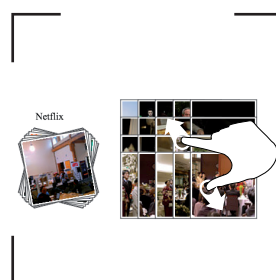


Abbildung 3.19:
Touch Stack Widget
nach [Murray, 2010]

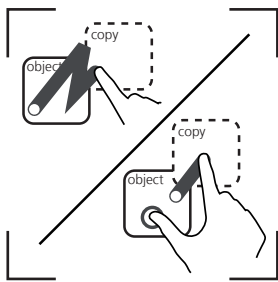


Abbildung 3.20:
Duplizieren durch
Abreißen (l.o.) und
Abziehen (r.u.)

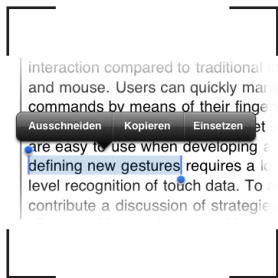


Abbildung 3.21:
Kontextmenü in Safari
unter iOS

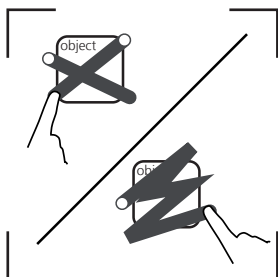


Abbildung 3.22:
Löschen durch
Durchkreuzen (l.o.)
oder Ausradieren (r.u.)

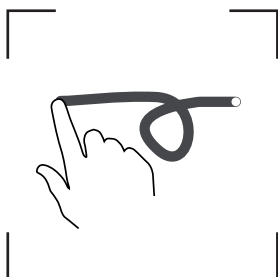


Abbildung 3.23:
Schleife nach links zum
Widerrufen von
Aktionen nach [Dorau,
2011, S. 160]

umgesetzt werden kann. (siehe Abbildung 3.19)

Kopieren und Einfügen

Dem Kopieren von Objekten steht das Duplizieren sehr nahe. Eine einfache Möglichkeit ist, das zu vervielfältigende Element mit einem Finger festzuhalten (HOLD) und mit einem zweiten die zu erwartende Kopie zu verschieben (DRAG). Nach [Dorau, 2011, S. 156] auch als *Abziehen* bezeichnet. Mit *Abreißen* benennt DORAU eine Geste, die durch eine schnelle Zick-Zack-Bewegung auf dem Objekt ein Duplikat erzeugt. Abbildung 3.20 illustriert die beiden Gesten. Das Einfügen eines Objektes wird üblicherweise mit einer DRAG&DROP-Geste realisiert, wobei entscheidend ist an welcher Stelle das Objekt losgelassen wird. Die Unterstützung dieser Anweisungen in Webanwendungen geht vom Browser aus und lehnt an klassische GUI-Konzepte an. Unter iOS wird basierend auf einer Selektion von Text oder anderen grafischen Elementen ein Kontextmenü aufgerufen, über das die Funktionen auswählbar sind. (siehe Abb. 3.21) Für den Aufruf eines Kontextmenüs hat sich die Geste mit der Bezeichnung TAP&HOLD durchgesetzt. [Dorau, 2011, S. 154] Dabei wird ein Finger auf dem Bedienelement, wie zum Beispiel ein Eingabefeld gehalten, während ein zweiter die Fläche kurz antippt. (vgl. Abb. 3.21)

Entfernen

Das Entfernen von Inhalten hat viele Facetten und beruht meist darauf, dass der Anwender die Rolle eines Gestalters einnimmt. Der Anwender erzeugt, ordnet an, und entfernt wenn nötig. Bezogen auf das Web finden sich diese Aufgaben in personalisierbaren Webportalen wie zum Beispiel iGoogle oder soziale Netzwerke wieder. Aber auch ins Internet ausgelagerte Werkzeuge wie Mail-Assistenten (beispielsweise Google Mail), Dokumenten-Sharing-Portale (beispielsweise Google Docs) oder Grafik- und Diagramm-Werkzeuge bis hin zu Online Betriebssystemen (OOS) unterstützen derartige gestalterische Prozesse, die das Entfernen von Objekten ermöglichen. Das einmalige *Durchstreichen* bietet sich vorallem für Text und Textbausteine an. Für größere Elemente oder Textpassagen wird das Durchkreuzen (siehe Abb. 3.22) bevorzugt. [Geißler, 2001, S. 92] Eine schnelle Zick-Zack-Bewegung (vgl. Abb. 3.22), die mit dem Ausradieren assoziiert werden kann, entspricht einer ausdrücklicheren Form des Entferns. (vgl. [Dorau, 2011, S. 158] und [Geißler, 2001, S. 92])

Undo und Redo

Undo und *Redo* beziehen sich als integrierte Browser-Funktion hauptsächlich auf die Folge besuchter Hypertext-Knoten, spich angesteuerter Web-Dokumente. Nach [Dorau, 2011, S. 160] kann die Anweisung mit Gestik umgesetzt werden, wenn man sich einen Zeitstrahl vorstellt und bildlich Vergangenheit (links) und Zukunft (rechts) trennt. Dementsprechend wird eine Schleife (CURL) nach links oder rechts ausgeführt, um in der Browser-History zu navigieren (vgl. Abb. 3.23). MOYLE ET AL. schlagen einen sehr ähnlichen Ansatz basierend auf *Marking Menus*⁶ vor und implementiert darauf aufbauend eine FLICK-Geste (erklärt unter dem Abschnitt "Scrollen und Blättern") in JavaScript. Die Evaluation zeigt, dass eine gestenbasierte Ausführung der Aktion gegenüber dem Betätigen des 'Back'-Buttons signifikant schneller ist (vgl. [Moyle und Cockburn, 2003]). Auf Marking Menus aufbauende Shortcut-Gesten sind ebenso für kontextbezogenen Funktion wie das Kopieren, Einfügen oder Löschen interessant.

⁶Marking Menus basieren auf dem Konzept von sogenannten *Pie-Menüs* - ein radiales Menü, dessen Segmente mit verschiedenen Kontextfunktionen belegt sind. Die Ausführungsrichtung der Geste bestimmt demnach die Aktion. Marking Menus verzichten auf diese grafische Unterstützung, womit der Fokus mehr auf der Ausführung selbst liegt. Das Prinzip ist jedoch das selbe. [Moyle und Cockburn, 2003]

Symbole

Symbolische Gesten finden sich in einem breiten domänenspezifischen Anwendungsfeld wieder und unterliegen kulturellen Unterschieden wie unterschiedliche Umgangsformen und Metaphern oder Unterschiede, die in der Sprache und dem Alphabet begründet liegen. Aus diesem Grund kommen symbolische Gesten wie auch Shortcuts nicht ohne Lernhilfen aus:

“As with other types of shortcuts, gesture shortcuts face several major challenges such as learnability, memorability and scalability.” [Li, 2010a]

Beispielhaft ist die Umsetzung des Kontextmenüs im Showcase von Moousture (vgl. Kapitel 3.3.1) zu nennen, das durch eine Kreis-Geste oder durch das Symbol ‘W’ aufgerufen wird. [Sibt-e-Hassan, 2010] Für die Selektion einzelner Objekte eignen sich Gesten wie das Ankreuzen [Geißler, 2001, S. 75] oder Abhaken [Geißler, 2001, S. 85]. Letzteres vor allem auch für die Bestätigung von Dialogen, wie es in Abbildung 3.24 gezeigt wird. [Geißler, 2001, S. 83] Ein Ausrufezeichen kann als eine nachdrückliche Aufforderung interpretiert werden oder das Zeichnen eines Fragezeichen als Hilfeaufruf [Wobbrock et al., 2009]. Weitere symbolische Gesten sind in [Wobbrock et al., 2007] und [Anthony und Wobbrock, 2010] aufgeführt.

Eine Besonderheit der direkten symbolbehafteten Eingaben ist die Handschrifterkennung. Das zugrundeliegende Alphabet beschreibt dabei die Ausführungsform der Symbole. Die komplexe Erkennung von Schreibschrift liegt dabei außerhalb des Fokus dieser Arbeit. Vielmehr geht es um das Potenzial von Gesten, einzelne alphabetische Symbole nachzubilden. Ein Anwendungsbeispiel zeigt das für Android-Systeme entwickelte Tool Gesture Search nach [Li, 2010a], mit dem es möglich ist eine Liste zu durchsuchen. Der Anfangsbuchstabe des gesuchten Listenelementes wird dazu über die gesamte Breite des Interface gezeichnet (siehe Abb. 3.25). Nichts desto trotz ist die Komplexität des lateinischen Alphabets nicht zu unterschätzen. GOLDBERG und RICHARDSON arbeiteten aus diesem Grund ein Symbolalphabet auf Basis von Einzelpfad-Gesten aus [Goldberg und Richardson, 1993], welches abgewandelt in der Gestenerkennungs-Software Graffiti unter Palm OS umgesetzt wurde. (vgl. [Blickenstorfer, 1995]) Abbildung 3.26 zeigt einen Auszug dieses sogenannten Uni-Stroke-Alphabets. Im Gegensatz dazu stellt POIRIER eine Eingabemethode vor, in der Buchstaben als Komposition einfacher Formen, wie Halbkreise und Linien, aufgefasst werden. [Poirier, 2005] Dieses Prinzip ist dem Formalisierungsansatz von GeForMT (vgl. Kapitel 3.3.2.3) im Grunde sehr ähnlich.

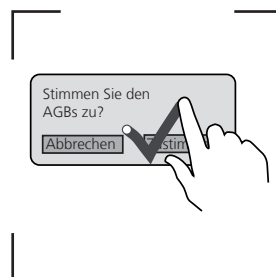


Abbildung 3.24:
Bestätigen in einem Dialog

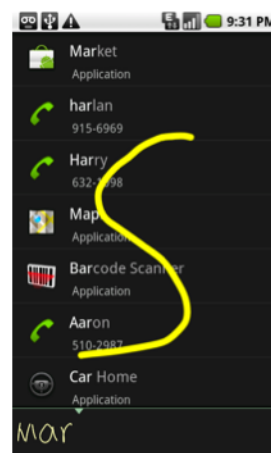


Abbildung 3.25:
Gesture Search [Li, 2010a]

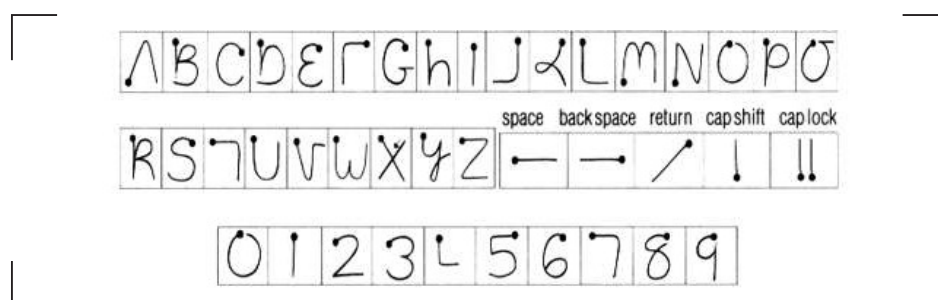


Abbildung 3.26: Ausschnitt aus dem Symbolalphabet in Graffiti [Blickenstorfer, 1995]

3.4.5 Bewertung des derzeitigen Stands touchbasierter Web-Anwendungen

Es ist festzustellen, dass sich die Entwicklung gestenbasierter Web-Anwendungen in einer Findungsphase befindet. Dies ist nicht zuletzt darauf zurückzuführen, dass die softwaretechnischen Lösungen noch sehr jung und nicht vollends ausgereift sind. (vgl. Kapitel 3.2.1) Die vorgestellten Anwendungsbeispiele dieses Kapitels weisen erste Prototypen auf, die eine Gesteninteraktion auf anwendungsspezifischen Web-Elementen in Betracht ziehen. Derartige Konzepte müssen sich jedoch noch in komplexeren Anwendungsbeispielen behaupten. Aktuell dominieren Gesten, die für die Steuerung von Web-Dokumenten wesentlich sind und vom Web-Browser bereitgestellt werden. Dazu zählen vor allem SHRINK und PINCH zum Skalieren und DRAG zum Verschieben des gesamten Dokumentes. (vgl. Kapitel 3.4.4) Diese adaptieren die Interaktionskonzepte von klassischen Fenstersystemen jedoch nur in einem gewissen Maß. Es wurde während der Recherche beobachtet, dass sich ein Großteil von Web-Anwendungen noch nicht auf die Domäne der mobilen Multitouch-Systeme eingestellt hat. In vielen Fällen sind Web-Implementierungen so sehr auf die Maus-Interaktion und größere Monitore spezialisiert, dass die Anwendung nicht über Tablets oder Smartphones bedient werden kann. Im umgekehrten Fall ist dieses Problem sogar noch stärker ausgeprägt. Bis auf einige wenige Ausnahmen wie zum Beispiel das in Kapitel 3.3.1 vorgestellte JavaScript-Framework *jQuery Mobile*, legen sich die Entwickler auf eine Optimierung für mobile berührungssensitive Eingabegeräte fest.

4 SYNTHESE

Dieses Kapitel unterteilt sich in zwei abgrenzbare Bereiche. In Kapitel 4.1 wird ein Szenario vorgestellt, das die Perspektiven von Web-Entwickler und Web-Nutzer näher bringt, die sich hinsichtlich der Anwendung eines JavaScript-Frameworks zur Gestenerkennung bieten. Kapitel 4.2 beschäftigt sich mit dem Thema der Gestenformalisierung und ihrer Integration in Web-Anwendungen.

4.1 ENTWICKLUNG UND ANWENDUNG EINER MOBILEN WEB-ANWENDUNG

Insbesondere mit der zunehmenden Mobilisierung von internetfähigen Geräten haben sich die Funktionen des Webs bedarfsgerecht und anwendungsspezifisch erweitert, dennoch ist die grundlegende Absicht der Nutzung des Internets - das Beziehen aktueller Informationen und neuen Wissens - unverändert und am verbreitetsten. Die Ergebnisse einer Nutzerstudie im mobilen Anwendungsbereich [Schmiedl et al., 2009] unterstreichen diese Tendenz.

“Based on a categorization of named websites 70% of the mobile accessed websites provide current (i.e. weather forecast, news, timetables) or general information (i.e. Wikipedia). 17% focus entertainment (Music, Video) and 13% Social Networks (i.e. Facebook)” [Schmiedl et al., 2009]

Obwohl ein auf Unterhaltung ausgerichtetes Anwendungsbeispiel, wie zum Beispiel ein Browser-Spiel, das größte Potenzial und besonders viel Spielraum für gestenbasierte Interaktionskonzepte bietet, konzentriert sich dieses Kapitel auf klassische, informative Webseiten, wie zum Beispiel ein Nachrichtenportal, Wissensdatenbanken wie www.wikipedia.de, aber auch einfache Onlinepräsentationen zur Vorstellung von Unternehmensleistungen. Ziel ist es, die gebräuchlichen Interaktionskonzepte des Webs für berührungsbasierte Eingabesysteme zu erweitern und zu optimieren, womit ein weites und profanes Anwendungsfeld für den Einsatz eines Web-Frameworks zur Gestenerkennung erschlossen werden soll.

Unter der festgelegten Fokussierung auf informative Web-Seiten, wird das Anwendungsszenario an dieser Stelle konkretisiert. Ausgangspunkt ist die Bedienung von Web-Anwendungen auf einem Multitouch-Tablet. Aufgrund des Formats eines Multitouch-Tablets, das der Größe eines Buches oder Schreibblocks gleichkommt, ist von der Bedienung mit einer Hand auszugehen während die andere das Gerät hält. Abbildung 4.1 illustriert das Szenario. In dem Fall, dass keine Eingaben auf

Intention der Nutzer

*Anwendungsbereich:
Informative Web-Seiten*



Abbildung 4.1:
Übliche Bedienung
eines
Multitouch-Tablets

dem Touchscreen erforderlich sind, umschließen beide Hände das Tablet für mehr Bequemlichkeit und Stabilität. (vgl. Abb. 4.2)

*Adaption des
Web-Interface*

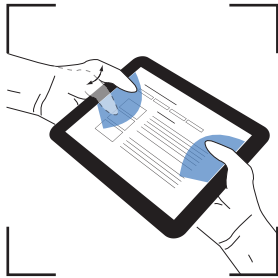


Abbildung 4.2:
Bequemlichkeitszonen

Wie in Kapitel 3.4.4 und 3.4.5 beschrieben wurde, implementieren mobile Web-Browser bereits Gesten, welche die Anwendung der ursprünglich für eine Maus-interaktion konzipierten Web-Seiten unterstützen. Oftmals müssen in diesem Fall jedoch Seitenbereiche mit interaktiven Element wie beispielsweise Links stark vergrößert werden, um sie bedienen zu können. Dabei geht jedoch auch der Überblick über das Web-Dokument verloren und eine Häufung dieser Notwendigkeit kann für den Nutzer frustrierend sein. Daher ist zu empfehlen, die Gestaltung mobiler Web-Anwendungen an die Erfordernisse des Gerätes anzupassen. Drei wichtige Aspekte zur Gestaltung von Multitouch-Anwendungen wurden in Kapitel 3.4.2 behandelt. Demnach ist darauf zu achten, dass interaktive Elemente groß genug und mit genügend Abstand zueinander gestaltet werden, damit sie nicht verfehlt oder verwechselt werden können. Softwaretechnisch kann die Adaption der Gestaltung mit Media-Queries realisiert werden, die in Kapitel 2.3.3 vorgestellt wurden. Um dem Gorilla-Arm-Problem entgegenzuwirken könnten wichtige Bedienelemente so angeordnet werden, dass weniger Bewegungen nötig sind. Abbildung 4.2 greift die stabilste Haltung des Tablet-Gerätes auf und veranschaulicht die ergonomisch nutzbare Fläche (blau markierter Bereich). In diesen seitlichen Bereichen, die in Anlehnung an die Interaktionszonen in [Henzen, 2010] auch Bequemlichkeitszonen genannt werden, können beispielsweise häufig angewendete Navigationsfunktionen wie das Blättern oder Scrollen der Seite umgesetzt werden. (siehe Kapitel 3.4.1)

Blättern und Scrollen

Das Scrollen in einem Web-Dokument ist ein Grundkonzept von Web-Browsern, das auch auf Multitouch-Geräten adaptiert wird. Insbesondere auf kleineren Displays gewinnt diese Funktion an Bedeutung. Aufbauend auf der implementierten Navigation in [Hunter et al., 2010], ist auf das Potenzial des Blätterns als Alternative oder zusätzliche Möglichkeit hinzuweisen. Web-Dokumente, die viel Text beinhalten und aus Kapiteln zusammensetzen könnten dadurch besser strukturiert werden. Darüber hinaus kann sich weniger Text auf einer Seite positiv auf das Leseverhalten des Anwenders auswirken. Einsatzmöglichkeiten des Konzepts zeigen sich überall dort, wo nummerierte Linkleisten implementiert werden, die auf weitere Einträge verweisen. Eine solche Navigationsleiste ist beispielsweise in www.google.de implementiert, wird aber auch häufig für Bildergalerien verwendet.

*Adaption von
Browser-Funktionen*

Ein weiterer Optimierungsvorschlag für die Gestaltung einer mobilen Web-Seite ist der Idee in [Moyle und Cockburn, 2003] nachempfunden und verbindet die Navigation in der Browser-History mit einer freien Geste. Hierfür eignet sich eine schleifenförmige Bewegung wie sie in Kapitel 3.4.4 vorgestellt wurde. In Folge der Übernahme häufig verwendeter Browser-Funktionen kann das Browser-Interface versteckt werden, was zu einem Platzgewinn für die Web-Inhalte führt.

*Symbolische Gesten als
Access Keys*

Das Potenzial von symbolischen Gesten wird vorrangig in der Adaption von Access Keys gesehen. Access Keys erlauben einen Schnelzugriff auf Seitenbereiche oder Menüpunkte und werden in der Regel über die Tastatur ausgeführt. (vgl. [W3C, 1999]) Beispielsweise könnten die Anfangsbuchstaben der Menüpunkte als Access Keys definiert werden. Übertragen auf ein gestenbasiertes Konzept, werden die Symbole als freie Gesten implementiert und können dementsprechend anwendungswelt ausgeführt werden. Von Vorteil ist diese Methode vor allem dann, wenn Links zu klein dargestellt sind oder zu dicht beieinander liegen, so dass sie aufgrund des Fat-Finger-Problems (vgl. Kapitel 3.4.2) nur schwer bedient werden können. In dem Fall wäre das Zeichnen des darauf verweisenden Symbols schneller und einfacher.

*Entwicklung mit
GeForMT*

Während eine lineare Geste für die Blättern-Funktion noch einfach zu realisieren ist, ist die Implementierung komplexer symbolischer Gesten mit einem kaum berechen-

baren Aufwand verbunden. Ein Web-Framework auf Basis von GeForMT (vgl. Kapitel 3.3.2.3) kann dem Web-Entwickler an dieser Stelle viel Arbeit abnehmen und wäre zudem flexibel einsetzbar. Aufbauend auf der Konzeption des Frameworks sollen die hier vorgestellten Anwendungsbeispiele in einem Prototypen umgesetzt werden. (siehe Kapitel 5.3.4)

4.2 WEBKONFORME GESTENFORMALISIERUNG

Auf Basis der GeForMT-Spezifikation, die in Kapitel 3.3.2.3 beschrieben wurde, stellt dieses Kapitel die Formalisierung von Gesten in den Kontext der Webentwicklung. Webkonform bedeutet in diesem Sinne sowohl syntaktische als auch semantische Konzepte der Sprache dahingehend zu adaptieren, dass sie mit standardisierten Webtechnologien einhergehen. Neben den zentralen Fragestellungen zur Integration der Gestenbeschreibung in Webanwendungen und der Objekt- und Aktionsbeschreibung einer Geste, werden auch allgemeingültige Konzepte der Sprache und dessen Erweiterungssponzial aufgegriffen und diskutiert.

4.2.1 Ereignisprofil atomarer Gesten

Das Ereignisprofil bildet die Grundlage für die Erkennung einer Geste. (vgl. Kapitel 3.1) Im Fokus dieser Arbeit liegt dabei auch die geräteunabhängige Beschreibung zweidimensionaler Gesten. Dies bedeutet für die softwaretechnologische Umsetzung eine Vereinheitlichung von Maus- und Touch-Events. Abbildung 4.3 zeigt die verschiedenen Ereignis-Typen und wie sie sich auf Basis der Erkenntnisse in Kapitel 3.4.1 aufeinander abbilden lassen. (vgl. Ereignis-Typen in Kapitel 2.2.2, Kapitel 2.3.6 und Kapitel 3.2.1)

*Abbildung von
Ereignis-Typen*

HIT	ENTER	MOVE	LEAVE	RELEASE
touchstart	touchenter	touchmove	touchleave	touchend
mousedown	mouseover	mousemove	mouseout	mouseup

Abbildung 4.3: Elemente des Ereignisprofils von Gesten

Pfadbasierte Gesten zeichnen sich durch die auf Bewegung basierenden Ereignisse ENTER, MOVE und LEAVE aus. Für atomare Gesten, die in GeForMT als LINE, CIRCLE, SEMICIRCLE und MOVE beschrieben sind, ist eine Häufung dieser Ereignisse charakteristisch. Entscheidend ist die Auswertung der Bewegungsform und -richtung. Statische Gesten kommen im Idealfall mit den Ereignis-Typen HIT und RELEASE aus, wobei Kontaktform und temporale Parameter in den Vordergrund rücken. Es können Gesten wie POINT und HOLD abgeleitet werden. DEPOINT ist ein Sonderfall, der als Kontaktverlust entsprechend dem RELEASE-Ereignis zu interpretieren ist und vorrangig als Trenner in der Definition zusammengesetzter Gesten dient.

*Nachbildung des
Ereignisprofils*

ENTER und LEAVE werden von der Sprache nach aktuellem Stand nicht erfasst. In Abschnitt 4.2.5 wird ihre Relevanz im Kontext des Objektbezugs diskutiert.

4.2.2 Formalisierung von Mausgesten

Voraussetzungen Aufbauend auf der Abbildung verschiedenartiger elementarer Ereignistypen im vorangegangenen Abschnitt ist die Gestenformalisierung bezüglich ihrer Beschreibungsfähigkeit von Mausgesten zu untersuchen. Ausgangspunkt ist, dass eine Mausgeste über das Feststellen der Maustaste definiert ist, um sich den Eigenschaften diskreter Berühreingaben anzunähern. (vgl. Kapitel 3.4.1) Das Gestenpotenzial der Maus kommt nicht an die Möglichkeiten von Multitouch-Gesten heran, ist unter den beschriebenen Voraussetzungen aber als eine Untermenge von Multitouch-Gesten beschreibbar.

Anwendbare Sprachbausteine Die Differenzierung bezieht sich hauptsächlich auf den Kontakttyp (siehe Abschnitt 4.2.3) und die Präsenz mehrerer Kontakte zur selben Zeit. Aus diesem Grund sind mehrere Regeln in GeForMT nicht auf Mausgesten anwendbar. (siehe grau hinterlegte Sprachbausteine in Listing 4.1) Das betrifft die Operationen für die synchrone und asynchrone Komposition von Gesten, sowie Relationen, die für die Spezifizierung synchroner Gestenzüge ausgelegt sind. Die Anzahl möglicher Kontaktpunkte ist im Funktions-Konstrukt entsprechend auf einen beschränkt. Die Restriktion führt die Sprache auf die Beschreibung von Single-Touch-Gesten zurück. Das Zeichen F für Berührungen mit dem Finger ist dieser Hinsicht eine vertretbare Formalisierung für einen Kontakt.

<code>relation</code>	<code>::= 'SYNC' 'JOIN' 'SPLIT' 'CROSS' 'CONNECT'</code>
<code>function</code>	<code>::= [integer] 'F' [integer] 'H' [integer] 'B'</code>
<code>operator</code>	<code>::= '*' '+' ',' ' '</code>
<code>integer</code>	<code>::= '1'</code>

Listing 4.1: Einschränkung der GeForMT-Syntax auf anwendbare Regeln für die Beschreibung von Maus- und Single-Touch-Gesten

4.2.3 Kontakttypen

Abstraktion und konkrete Ausprägung Das Architekturmodell (vgl. Kapitel 3.1) sieht eine Abstraktion der Eingabedaten auf unterster Ebene vor. Damit ist die Spezifizierung primitiver Eingaben auf Anwendungsebene weitestgehend unabhängig von der eingesetzten Technologie, da das System die Daten im einfachsten Fall in diskrete Werte zur Positionsbestimmung übersetzt. Das standardisierte Touch-Event-Interface sieht die Bereitstellung von Information zu den Maßen, der Ausrichtung und dem Druck einer Berührung vor, sofern die Parameter durch entsprechende sensorische Vorrichtungen messbar sind. (vgl. Kapitel 3.2.1)

Unterscheidung Die Erkennung der Hand oder eines Blobs, wie sie in GeForMT als Sprachoptionen H und B angedacht sind, ist bisher kein verbreitetes Bedienkonzept in Webanwendungen. Dies ist darauf zurückzuführen, dass die Multitouch-Technologie vorrangig in mobilen Endgeräten Anwendung findet. Deren Interaktionskonzepte konzentrieren sich auf Eingaben mit dem Finger oder einem beigefügten Stift (Stylus). Theoretisch kann auf Basis der Größe und Form des Kontaktpunktes auf den Kontakttyp geschlossen werden und die Auswahl anwendbarer Gesten gegenüber dem Eingabegeräte adaptiert werden. Für die Klassifikation kommen der Kontakt durch einen Maus-Cursor, der Stiftkontakt und die Berührung mit dem Finger in Frage.

Die Einteilung bietet das Potenzial Gesten an einen anwendungsspezifischen Eingabemodus zu knüpfen, der sich aus der Wahl der Eingabetechnik ergibt. Beispielsweise ein Schreibmodus bei Erkennung einer Stifteingabe. Gegen eine derartigen Klassifikation im Sprachkonzept von GeForMT spricht allerdings die Anforderung an eine geräteunabhängige Verwendung des Frameworks, in der die Funktionsfähigkeit einer Webanwendung ungeachtet der Eingabetechnik gewährleistet sein soll. Es wird demnach eine Verallgemeinerung des Kontakttyps bevorzugt, wie er in Abschnitt 4.2.2 definiert wird.

Vor- und Nachteile der Klassifikation

4.2.4 Sequentielle Komposition

Sequentielle Verknüpfungen atomarer Gesten werden in GeForMT mit dem Komma-Operator spezifiziert. Dabei ist jedoch nicht festgelegt, ob der Kontakt dabei verloren gehen darf. Hinsichtlich einer Klassifikation zwischen Einzelpfad-Gesten und zusammengesetzten Gesten (vgl. Kapitel 2.2.3) ist eine Erweiterung der Sprache notwendig. Syntaktisch bietet sich das Semikolon als ausdrucksstarkes Trennzeichen zwischen Einzelpfaden an. Die EBNF-Regel **operator** wäre demnach wie folgt zu erweitern:

Komposition mit dem Komma-Operator

```
operator ::= ' | ' | '*' | '+' | ',' | ';' ;
```

Listing 4.2: Angepasste EBNF-Regel für Komposition von Gesten

Mit Einführung des Verknüpfungsparameters **CONNECT** wurde eine Relation eingeführt, die eine genauere Lagebeschreibung ermöglicht. Er definiert nach [Henzen, 2010], dass die Endposition eines Gestenzuges die Anfangsposition einer nachfolgenden Geste ist. Die Definition der Verknüpfungs-Operatoren ist wie folgt zu konkretisieren:

*Verknüpfungsparameter
CONNECT*

Das Komma definiert eine sequentielle Verknüpfung von Bewegungspfaden, die im Rahmen eines Gestenzuges ausgeführt werden, ohne dass der Kontakt verloren geht. Die Operation schließt implizit die Bedingungen der **CONNECT**-Relation mit ein. Das Semikolon legt hingegen fest, dass der Gestenzug endet (**RELEASE**) und neu beginnt (**HIT**). Die Lage des darauffolgenden Gestenzuges ist nicht durch die Operation festgelegt.

Definition: Komma und Semikolon

Mit dem Schlüsselwort **CONNECT** existiert aktuell nur eine einzige Möglichkeit, die Lage einer anknüpfenden atomaren Geste zu beschreiben. Ein anderer Lösungsweg der Lagebeschreibung wird in [Henzen, 2010] mit der Verschachtelung von Gestenatomen unter Angabe der Richtung vorgeschlagen. Dieser Ansatz erhöht die Komplexität der Sprache jedoch erheblich. Aus diesem Grund werden im folgenden erweiternde Sprachbausteine vorgeschlagen, die sich in den Bestand der Relationen einfügen und die relative Lage zusammengesetzter Gesten bei minimaler Veränderung der Syntax konkretisieren. Die Ansätze sind in Tabelle 4.1 zusammengefasst und werden in der beispielhaften Skizzierung in Abbildung 4.4 veranschaulicht.

Räumlichen Anordnung zusammengesetzter Gesten

Der erste Vorschlag betrifft die Erweiterung von **CONNECT** um einen Parameter, der den Ort der Verknüpfung festlegt - die Anfangsposition oder Endposition des vorangegangenen Gestenzuges. Während **CONNECT_END** der Definition der **CONNECT**-Relation nach [Henzen, 2010] entspricht, wird mit **CONNECT_START** die Bedingung eingeführt, dass alle Gestenzüge im gleichen Punkt starten. (siehe Abb. 4.4a)

Anknüpfung

Ein zweiter Ansatz ist die Klassifikation von horizontal und vertikal angeordneten Gesten. Durch die Relationen **AMONG** und **ASIDE** lassen sich diese Bedingungen formal

Horizontale und vertikale Nachbarschaft

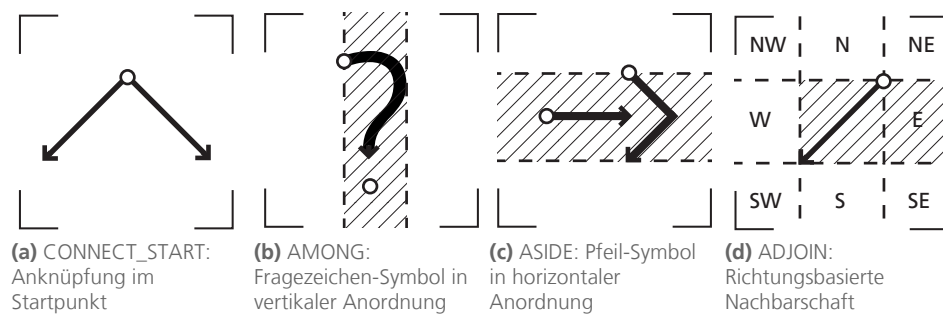


Abbildung 4.4: Beispiele für die Formalisierung der räumlichen Anordnung zusammengesetzter Gesten (Legende: *Schraffierte Bereiche verdeutlichen den Gültigkeitsbereich von verknüpften Gestenzügen*)

Entsprechende GeForMT-Ausdrücke:

- a) CONNECT_START[LINE_SW; LINE_SE]
- b) AMONG[SEMICIRCLE_S_CW, LINE_S; POINT]
- c) ASIDE[LINE_E; LINE_SE, LINE_SW]
- d) ADJOIN_E[LINE_SW; ..]

beschreiben. Abbildung 4.4b zeigt die gestenhafte Skizzierung eines Fragezeichens, dessen Gültigkeitsbereich sich über ein horizontales Band erstreckt. Abbildung 4.4c zeigt den Fall einer horizontalen Anordnung mit einer Pfeil-Geste. Die Schwäche dieses Ansatzes ist jedoch, dass keine konkrete Richtung der Anordnung festgelegt ist. Der Punkt des Fragezeichens könnte genauso auch über den ersten Gestenzug platziert sein.

Nachbarschaft mit optionaler Richtungsangabe

Ein dritter Ansatz berücksichtigt die Richtung basierend auf der EBNF-Regel *direction* (vgl. Kapitel 3.3.2.3). Abbildung 4.4d stellt einen ersten Gestenzug dar, dessen Umfeld in acht mögliche Quadranten unterteilt ist, in denen ein weiterer Gestenzug ausgeführt werden kann. Beispielsweise könnte mit Angabe des Relations-Ausdrucks **ADJOIN_E** die gerichtete Anordnung (von links nach rechts) der Pfeil-Geste aus Abbildung 4.4c konkretisiert werden.

Ansätze der räumlichen Anordnung	Schlüsselwörter
Anknüpfung	CONNECT_START CONNECT_END
Horizontale und vertikale Nachbarschaft	AMONG ASIDE
Nachbarschaft mit optionaler Richtungsangabe	ADJOIN[_direction]

Tabelle 4.1: Zusammenfassung der Ansätze zur Beschreibung der räumlichen Anordnung zusammengesetzter Gesten

Uneindeutige Beschreibung der Bewegungsform

Durch die Unterscheidung geometrischer Formen und Bewegungsrichtungen in GeForMT können bereits viele Bewegungsformen formal beschrieben werden. Dabei gibt der Formalisierungsansatz jedoch nicht die räumliche Dimension einer atomaren Geste vor. Die Länge einer Linie oder der Radius eines Kreises können variieren. Das folgende Beispiel zeigt die Uneindeutigkeit einer Beschreibung, die hierbei entstehen kann. Eine bedeutende Bewegungsform ist neben der Kreis-Bewegung die spiralförmige Bewegungen. (vgl. Kapitel 2.2.3) Abbildung 4.5 stellt die besagte Bewegungsform dar, die sich in GeForMT-Syntax wie folgt beschreiben lässt:

`1F(SEMICIRCLE_S, SEMICIRCLE_N, SEMICIRCLE_S).`

Der Ausdruck könnte aber auch als geschlossener Kreis oder als invertierte Spiralförmigkeit interpretiert werden.

Angelehnt an den Lösungsansatz in der Sprache GDL, beschrieben in Kapitel 3.3.2.1, wird eine weitere Relation mit der Bezeichnung `CLOSE` vorgeschlagen. Sie definiert, dass der Bewegungspfad eine geschlossene Form ergeben muss. Folglich entspricht die Anfangsposition der Geste der Position des Endpunktes. Die Größenverhältnisse werden mit dem Konstrukt zwar nicht explizit bestimmt, es schränkt den Interpretationsspielraum jedoch erheblich ein. Abbildung 4.6 stellt ein geschlossenes Rechteck dar, das wie folgt zu definieren wäre:

`CLOSE[LINE_E, LINE_S, LINE_W, LINE_N].`

4.2.5 Geltungsbereich und Objektkontext

Der Kontext einer Geste ergibt sich in erster Linie durch den Ausführungsort auf der Benutzeroberfläche. Liegt dieser im definierten Geltungsbereich der Geste, sind in zweiter Instanz deren Ausführungseigenschaften zu analysieren. Der Geltungsbereich einer Geste ist durch das Größenformat zugeordneten UI-Elements bestimmt. Bezugnehmend auf das zu Grunde liegende Box-Modell eines Web-Elements (vgl. Kapitel 2.3.3) erstreckt sich der Bereich über die *Contentbox* und die umgebenden Boxen *Padding* und *Border*.

Die User-Interface-Komponenten der Webanwendung werden in der Regel im HTML-Dokument definiert. Sie liegen dann als hierarchisch angeordnete Objekte im Speicher vor und sind über die DOM-API mindestens über ihren HTML-Tagnamen identifizierbar. Eine eindeutige Auszeichnung bietet das Attribut `id`, wie es in *WKTou* (vgl. Kapitel 3.3.1) zur Herstellung des Objektbezugs verwendet wird.

Eine naheliegende Lösung ist die Adaption von Selektoren, wie sie standardmäßig in CSS-Regeln für die Zuweisung von Stilregeln verwendet werden. (siehe Kapitel 2.3.3) Ihre Syntax ist dem Web-Entwickler im Allgemeinen bekannt und leicht erlernbar. Syntaktisch sind Selektoren und GeForMT gut zu vereinen. Dies erlaubt eine nahtlose Integration in die Gestenbeschreibung. Beispielsweise definiert der Ausdruck `1F(MOVE(img#picId))` eine Bewegung auf einem Bild mit dem Id-Attributwert `picId`. Die entsprechende Adaptierung der GeForMT-Grammatik ist in Listing 4.3 aufgeführt. Von der Abbildung der komplexen Struktur eines Selektor-Ausdrucks wird dabei abgesehen. Die EBNF-Regel *focus* grenzt lediglich den gültigen Zeichenvorrat ein.

```
atomfocus ::= atom | atom('focuslist')
focuslist ::= focus ('', 'focus')*
focus      ::= [A-Za-z0-9#.:!'"^$*~()[]|\s-~]+
```

Listing 4.3: Angepasste EBNF-Regeln für die Nutzung von Selektoren als Objektidentifikation

Das Konzept ermöglicht die Bestimmung von Objektmengen durch die Vergabe von Klassen-Selektoren, Auzählungen oder ähnliche Ausdrücke, wie sie in Kapitel 4.3 vorgestellt wurden. Die Regel *focuslist* stellt konsequenterweise eine redundante Funktion dar. Die Auswahl von Objekten durch Selektoren ist gegenüber der ursprünglichen Idee hinter dem *focuslist*-Konstrukt jedoch anders zu interpretieren. In CSS beinhaltet der Typ-Selektor beispielsweise eine Anwendung der Eigenschaften

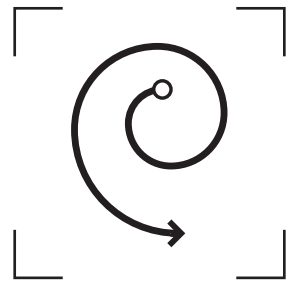


Abbildung 4.5:
Spiralförmige
Bewegung

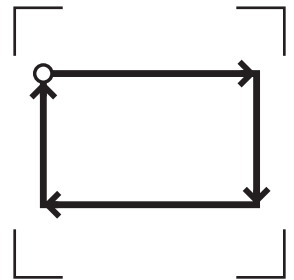


Abbildung 4.6:
Geschlossenes Rechteck

*Identifikation von
Elementen des
User-Interface*

*Formalisierung des
Objektbezugs*

Objektmengen

auf jedes Element von diesem Typ. Übertragen auf die Formalisierung ist die Geste auf verschiedenen Objekten auf gleiche Weise anwendbar und kommt somit einer Art Vorlage für eine Gruppe von Elementen gleich. Beispielsweise definiert der Ausdruck $1F(MOVE(img))$, dass alle Bilder der Webseite verschoben werden können. Die ursprüngliche Bedeutung einer Objektmenge ist aber mit dem Stichwort 'Umfassen' zu beschreiben. Das bedeutet, dass der Bezug zu jedem Objekt der Menge hergestellt sein muss. Ein Beispiel ist die LASSO-Geste, die eine Objektmenge umschließt. Das Beispiel wird in den nächsten Absätzen noch einmal aufgegriffen. Die Arbeit stützt sich im weiteren auf die erste der beiden Bedeutungen, welche sich hinsichtlich der verwendeten Web-Technologien als der praktikablere Ansatz erweist. In weiterführenden Arbeiten könnte eine differenziertere Unterscheidung anhand von Selektor-Typen diskutiert werden.

Optionale Angabe

Gemäß dem Fall, dass kein Fokus festgelegt ist (Beispiel: $1F(MOVE)$), kann davon ausgegangen werden, dass es sich um eine freie Geste handelt, die anwendungsweit gültig ist. Eine geeignete Lösung ist, die Geste an das Struktur-Element `html` zu binden.

Elementstruktur

Das Box-Modell ist ein wichtiges Konzept für die Gestaltung des Webinterface, da es auf jede UI-Komponente angewendet werden kann. Bis auf die Begrenzung eines interaktiven Geltungsbereichs, hat die Box-Einteilung jedoch keine Bedeutung für die Interaktion, sondern ist ein rein gestalterisches Mittel. Die Webstandards sehen in ihren Schnittstellen nach derzeitigem Stand keine Unterscheidung vor, der erkennen lässt, ob sich der Ausführungsort einer Geste auf dem Rand, im Padding-Bereich oder zentral auf dem Inhalt befindet. Eine daran angelehnte Formalisierung von Interaktionsbereichen kann jedoch für die Erkennung der Intention einer Anweisung wertvoll sein. Sie ermöglicht weitere kontextabhängige Funktionen, insbesondere spezifischere Option der direkten Manipulation.

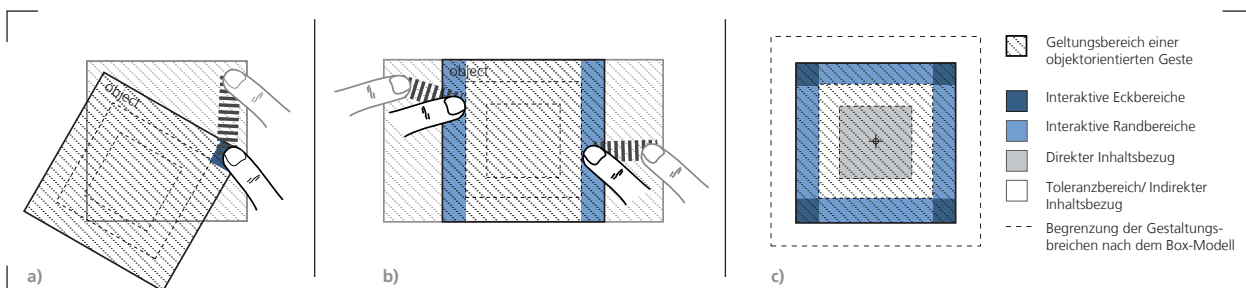


Abbildung 4.7: Formalisierung von Interaktionsbereichen zur Spezifizierung des Objektkontexts:

- a) Translatorische Bewegung auf einem Eckbereich
- b) Skalierung eines Objektes in horizontaler Richtung über die Randbereiche
- c) Allgemeine Unterteilung in interaktive Bereiche

Beispiele

Beispielsweise ist für eine physische Geste (Beispiel: $1F(MOVE(\#object))$), die nach Definition (vgl. Kapitel 2.2.3) ein realistisches Reaktionsverhalten anstrebt, bedeutend, ob sich der Angriffspunkt mittig oder auf einer Ecke des Elements befindet. (vgl. Abb. 4.7a) So kann aus dem Kontext bereits die Art der Manipulation geschlossen werden. Zum Beispiel, ob das Objekt nur verschoben oder auch gedreht werden soll. Bezüglich der Objektskalierung könnte die Differenzierung von Ecken und Kanten Möglichkeiten für die Einstellung der Proportionserhaltung bieten. Beispielsweise kann die Fokussierung gegenüberliegender Randbereiche als Skalierung in horizontaler oder vertikaler Richtung (vgl. Abb. 4.7b) interpretiert werden, während die Interaktion auf Eckbereichen für eine proportionale Skalierung reserviert ist. Aus der

Kombinierung ergibt sich weiteres Potenzial. Zum Beispiel das Festhalten des Inhaltes im Zentrum und Agieren auf den Randflächen. Daraus folgt eine Einteilung des Objektkontextes in interaktive Eck-, Rand- und Kernflächen, wie die Abbildung 4.7c grafisch veranschaulicht.

Im Mittelpunkt steht die Information, zum Beispiel eine Grafik oder Text. Handlung und Information stehen hier direkt in Beziehung. Die Problematik der Verdeckung (vgl. Kapitel 3.4.2) ist hier jedoch besonders signifikant und sollte bei der Wahl der Gesten berücksichtigt werden. Der umgebene Toleranzbereich kann diesbezüglich Abhilfe schaffen. Er ist der *padding*-Eigenschaft des Box-Modells nachempfunden und grenzt den Inhalt deutlich vom Rahmen des Objekts ab. Der Rahmen stellt als äußerste Grenze des Elements eine Beziehung zur Umwelt her. Ein Raster teilt den Rahmen in interaktive Eck- und Randflächen. Das Box-Modell dient hier lediglich als Vorbild und beschränkt sich nicht auf eine derartige softwaretechnische Implementierung. Die Verschachtelung von Strukturelementen, die jeweils einen eigenen Geltungsbereich definieren, könnte ebenso ein Ansatz für die Realisierung sein.

Formalisierung von Interaktionsbereichen

Im einfachsten Fall wird der Erkennungsbereich des Objekts nicht verlassen, wodurch der Objektbezug fortwährend hergestellt werden kann. Einige Gesten werden jedoch im Abseits des Erkennungsrahmens ausgeführt und sind dennoch objektbezogen, wie zum Beispiel das Durchkreuzen (STRIKE) eines Objekts (Abb. 4.8a), um es zu entfernen oder eine LASSO-Geste um ein oder mehrere Objekte zu selektieren (Abb. 4.8c und Abb. 4.8e).

Eintreten und Verlassen des Erkennungsbereich

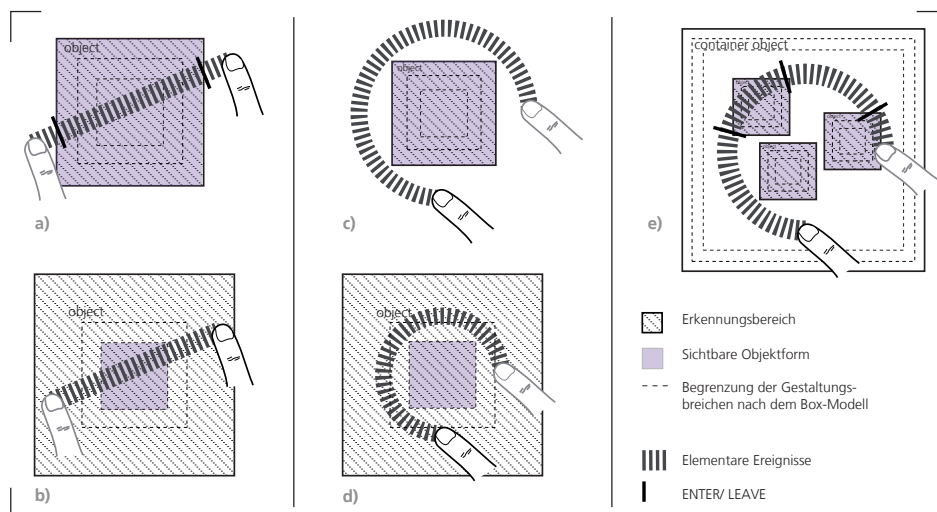


Abbildung 4.8: Herstellung des Objektbezugs von Gesten außerhalb des Erkennungsbereichs

Naheliegende Formalisierungsansätze:

- a) `1F(ENTER(#object), LINE(#object), LEAVE(#object))`
- b) `LINE(#object)`
- c) `CIRCLE` (ohne Objektbezug)
- d) `CIRCLE(#object)`
- e) an Ausführungsform orientiert: `CIRCLE(#container)`
oder ereignisorientiert: `ENTER(.objects)`

Erster Anhaltspunkt für die Auflösung dieser Problematik ist das Ereignisprofil bezüglich ENTER- und RELEASE-Ereignissen auszuwerten. Für die STRIKE-Geste ist die Ereigniskette ENTER-MOVE-LEAVE charakteristisch. Start- und Endpunkt werden nicht erfasst. Eine naheliegende Lösung, um diese Geste formal auszudrücken, wäre die Erweiterung der atomaren GeForMT-Gesten um entsprechende, gleichnamige Sprach-

bausteine. (siehe Beschriftung von Abb. 4.8a) Dieser ereignisorientierte Ansatz eignet sich auch für die Selektion von Objektmengen, basierend auf der Angabe eines Klassenselektors (siehe Beschriftung von Abb. 4.8e). Er ist jedoch nicht für eine LASSO-Geste praktikabel, die Objekte umschließt, ohne sie zu treffen. (vgl. Abb. 4.8c) Eine alternative Lösung, die die formale Spezifikation unberührt lässt, ist die Erweiterung des Erkennungsbereichs. Im Fall eines einzelnen Objekts wird der Randbereich zum transparenten Toleranzbereich. Visuell liegt der Ausführungspfad außerhalb der Objektgrenzen. Der Objektbezug ist dennoch gegeben. (vgl. Abb. 4.8b und Abb. 4.8d) Bei Objektmengen ist hingegen ein übergeordneter Erkennungsbereich erforderlich, der durch das *Container*-Objekt gegeben ist. Diese Lösung bezieht sich auf die Ausführungsform. Die Selektion umschlossener Objekte ist dann aber anwendungsspezifisch nach dem Erkennungsprozess vorzunehmen.

4.2.6 Aktionsbeschreibung und -zuordnung

Die Aktion steht als zentrale Komponente der Gesteninteraktion in Relation zum Kontext (Objekt) und der formal beschriebenen Bewegung (Geste). (vgl. Abb. 3.11 in Kapitel 3.3.2.3) Sie beschreibt das Verhalten der Anwendung im Fall der Erkennung einer Geste. Das Framework muss in der Lage sein diese Beziehungen herzustellen. Dies setzt eine konkrete Schnittstellen-Definition sowohl auf Seite der formalen Sprache als auch auf Framework- und Anwendungsebene voraus. Im Kontext der JavaScript-Programmierung bietet sich für die Implementierung von Aktionen das Konzept der Ereignisbehandlung an. (siehe Kapitel 2.3.6)

*Aktionszuordnung als
Bestandteil der Sprache
GeForMT*

Ein erster Ansatz, um die Beziehung der Interaktions-Komponenten herzustellen, ist die Abbildung von Objekt-Klassen und Methoden als Bestandteil der formalen Definition in GeForMT. (vgl. Abb. 3.11 in Kapitel 3.3.2.3) Listing 4.4 zeigt, wie das Sprachkonzept folglich unter Berücksichtigung der Eigenschaften von JavaScript zu erweitern ist.

```
definition ::= complex ('|' complex)* ['_time'] ['->' handler]
handler    ::= [A-Za-z0-9_-]+ ('.' [A-Za-z0-9_-]+)* ['()']
```

Listing 4.4: Angepasste EBNF-Regel für die Aktionszuweisung angelehnt an [Henzen, 2010, S. 83]

Der Sprachzusatz spezifiziert den Eventbezug, indem der Formalisierung der Bezeichner des zugehörigen Event-Handlers angehangen wird. Im Sinne der Objektorientierung wird dabei auch die Möglichkeit berücksichtigt den Geltungsbereich des Handlers anzugeben. Runde Klammern am Ende sind lediglich unterstützende Elemente, um den Funktionscharakter des Konstrukts zu verdeutlichen. Der Ausdruck **draw=MOVE->DrawContext.draw()** ordnet der Geste beispielsweise die Handler-Funktion **draw** zu, die im anwendungsspezifischen Objekt **DrawContext** implementiert ist.

*Aktionszuordnung
durch Objektmodell*

Ein anderer Ansatz basiert auf der Zuordnung durch eine Datenstruktur, die der Beschreibungssprache übergeordnet ist. In JavaScript bieten sich hierfür Objekt-Literale an. In Kapitel 2.3.5 wurde beschrieben, dass Objekt-Literale für die Konfiguration verwendet werden. Im folgenden wird daher auch von der *Konfiguration einer Geste* gesprochen. GeForMT wird dabei zu ihrer wesentlichen Aufgabe zurückgeführt, indem die Sprachbausteine auf die Formalisierung der Bewegung reduziert wird. Der Bezeichner und die Aktionsbeschreibung werden außerhalb der Beschreibungssprache definiert. Die Beziehungen bleiben durch die Objektstruktur gewahrt. Listing

4.5 greift das Beispiel des vorangegangenen Absatzes auf, um den Ansatz zu erläutern.

```

1 var draw = {
2   identifier: "draw",
3   expr : "MOVE",
4   handler: function(e){ .. } // oder auch: DrawContext.draw
5 }

```

Listing 4.5: Aktionszuweisung basierend auf einem Objekt-Literal in JavaScript

Der Name der Geste wird mit dem Variablenbezeichner des Literals definiert, kann aber vereinfacht unter **identifier** festgelegt werden. Das Schlüsselwort **expr** weist den GeForMT-Ausdruck zu. Die Variable **handler** beinhaltet die Handler-Funktion. Die Implementierung der Funktion kann direkt im Objekt-Literal vorgenommen werden. Möglich ist aber auch ein Verweis auf die Funktion, ähnlich wie im sprachbasierten Ansatz. Dies ermöglicht dem Webentwickler frei zu entscheiden, an welcher Stelle des Programms er die Folgeaktion implementiert.

Gegenüber der Aktionszuweisung als Bestandteil von GeForMT hat der zweite Ansatz den Vorteil, dass die Implementierung (Aktionsbeschreibung) direkt zugeordnet werden kann. Nachteilig ist jedoch der höhere Lernaufwand, der mit der zusätzlichen Spezifikation eines Objektmodells einhergeht, denn die Schnittstelle erfordert weiteres Implementierungswissen über den Aufbau und zu verwendende Bezeichnungen des Objekt-Literals. Der erste Ansatz ist in dieser Hinsicht die einfachere Methode, da sie sich durch eine kompakte Schreibweise und mehr Flexibilität auszeichnet. Die Übersichtlichkeit und Zuordenbarkeit ist jedoch diskutabel. Der zweite Ansatz stellt sich als praktikabel heraus, wenn es um die Erweiterbarkeit und Wiederverwendung von Gestendefinitionen geht. In Kapitel 4.2.7 wird das Potential eines Konfigurations-Objektes nochmals aufgegriffen.

Vor- und Nachteile

4.2.7 Konfigurationsparameter

Im vorangegangenen Abschnitt wird mit dem Ansatz eines Objekt-Literals nahe gelegt, die Gestenbeschreibung in Form eines GeForMT-Ausdrucks von der framework-spezifischen Konstruktion zu separieren. Es wird demnach der Standpunkt vertreten, dass die Beschreibungssprache keine softwaretechnischen Konzepte abbilden sollte (zum Beispiel eine Handler-Funktion, vgl. Abschnitt 4.2.6), die die Interoperabilität auf eine bestimmte Programmiersprache oder ein bestimmtes Gestenerkennungsverfahren einschränken. Aus diesem Blickwinkel ist auch die Verwendung von Selektoren für den Objektbezug (vgl. Abschnitt 4.2.5) in weiterführenden Arbeiten zu überdenken und gegebenenfalls zu abstrahieren.

Interoperabilität

Als ebenso nachteilig wird die Erweiterung von Sprachbausteinen gesehen, die das Framework konfigurieren. Ein Beispiel hierfür ist die Einführung der Sprachbausteine *ON* und *OFF* (vgl. Kapitel 3.3.2.3), um die Strategien der Gestenerkennung zu bestimmen. Dieser Parameter wird als framework-spezifische Konfiguration angesehen. Mit der Separierung wird das Ziel verfolgt, einer zunehmenden Spezialisierung der Sprachbestandteile auf einen bestimmten Anwendungsbereich entgegenzuwirken. Beispielsweise könnte die Sprache im Bereich der Visualisierung von Gesten eingesetzt werden, insbesondere in der Umsetzung von Hilfesystemen (Feed-Forward). Auch Entwicklungswerkzeuge könnten von einer Generalisierung profitieren. Die

Verwendung von GeForMT als Austauschformat kann sich an der Stelle als Vorteil erweisen.

Erweiterbarkeit Ein weiteres Ziel ist, die Konfiguration einer Geste erweiterbar zu gestalten. Bezugnehmend auf die in Kapitel 3.3.2.1 und 3.3.2.2 vorgestellten Ansätze in TouchToolkit und Midas, könnten zeitliche und räumliche Einschränkungen vorgenommen werden, um die Geste zu spezifizieren. In Midas ist es beispielsweise möglich einen zeitlichen Rahmen festzulegen. (vgl. Kapitel 3.3.2.2) Die Beschreibungssprache in TouchToolkit bietet beispielsweise Ausdrücke an, um Grenzwerte der Pfadlänge oder der Größe der eingeschlossenen Fläche festzulegen. (vgl. Kapitel 3.3.2.2) Weiterhin könnten Einstellungen bezüglich der Geschwindigkeit oder dem auszuübenden Druck für die Gestenerkennung von Bedeutung sein. Interessant erscheint vor allem eine Konfiguration, in der die Toleranz bei der Interpretation einer Geste angepasst werden kann. Beispielsweise, indem eingestellt wird, wie viel Prozent Übereinstimmung mindestens notwendig ist, dass die Geste als erkannt gilt. Die Konfigurationsparameter würden letztlich als Variablen des Objekt-Literals implementiert. (vgl. Listing 4.5)

4.2.8 Potenzial der Wiederverwendung von Definitionen

Wiederverwendung Die Wiederverwendung von Definitionen ist von Vorteil, wenn sich Teilausdrücke wiederholen oder Bestandteil komplexerer Gesten sind. Für die Wiederverwendung wird der eindeutige Bezeichner einer Geste dazu benutzt, um ihn in einer anderen Definition einzubetten. (vgl. [Henzen, 2010])

Einschränkungen Zu beachten ist dabei, dass der Bezeichner einer Geste nicht im dazugehörigen GeForMT-Ausdruck enthalten ist, da mit dem Ersetzungsprozess des Frameworks eine Rekursion durchläuft, die nur durch eine eingreifende Abbruchbedingung terminiert. Man könnte sich diese rekursive Vorschrift zu nutze machen, allerdings sind hierfür noch Anwendungsbeispiele zu finden. Desweiteren ist zu prüfen, dass der Ausdruck der Geste auch nach der Substitution des Bezeichners ein gültiger Ausdruck gemäß der GeForMT-Syntax ist. In der Implementierung der Wiederverwendung zeigt sich der Bedarf von Symbolen, die den Gesten-Bezeichner eindeutig vom Rest des Ausdrucks abgrenzen. Dies zeichnet das Nicht-Terminalsymbol zum einen auf geeignete Weise aus. Zum anderen wirkt es dem Fall entgegen, dass Bezeichner in anderen Bezeichnern enthalten sein können. Die Einklammerung in spitzen Klammern, wie in Listing 4.6 vorgestellt wird, erscheint hierfür als geeignet. Es wird dazu das Beispiel aus [Henzen, 2010] aufgegriffen, das einen Pfeil Richtung Norden und dessen zweimalige sequentielle Ausführung beschreibt. Die Freiheiten in der Wahl des Bezeichners bleiben durch die Abgrenzung gewahrt.

```
Nordpfeil = LINE_N;LINE_NE,LINE_SE
Doppelpfeil = <Nordpfeil>;<Nordpfeil>
```

Listing 4.6: Klammerung von wiederverwendeten Bezeichnern

Wiederverwendung durch Beschreibungs-vorlagen Eine andere Form der Wiederverwendung ist in Abschnitt 4.2.5 mit der Definition von Objektmengen mit Selektoren erklärt. Im Folgenden wird ein alternativer Ansatz vorgestellt, in dem die Beschreibung der Gestenausführung als objektunabhängige Vorlage verwendet werden kann.

Der *Aktion*, die durch die Erkennung einer Geste ausgeführt wird, steht die *Anweisung* gegenüber, wie es in Kapitel 3.4.3 erklärt wurde. Die Gestenbeschreibung

mit GeForMT sind hauptsächlich in das dort erläuterte Abstraktionslevel *gestural* einzuordnen, da sie die Handlung selbst abbildet. Die Erweiterung des Wiederverwendungskonzepts in der Form wie sie in Listing 4.7 beispielhaft abgebildet ist, ermöglicht die Formalisierung von Gesten auf der *representational*-Ebene. Der Objektkontext wird in der Beschreibung offen gehalten, indem stattdessen Platzhalter definiert werden. (vgl. Listing 4.7, Z. 2) Dieses Konstrukt bildet die Beschreibungsvorlage, die lediglich die Bewegungsform der Geste abbildet. Die Anwendung der Vorlage hat den Charakter einer Funktion, die die Gestenbeschreibung in einen Kontext setzt, der sich hauptsächlich über den Geltungsbereich definiert. Als Platzhalter bieten sich $\$$ -Marken an, gefolgt von einem Zahlenwert, der die Parameterposition in der Anwendungsfunktion repräsentiert.

```
// Gestenbeschreibung als Vorlage (gestural)
pinch = SPLIT[LINE($1)*LINE($1)]
// Geste mit Objektbezug (representational)
scale = pinch(#id)
```

Listing 4.7: Verwendung von Beschreibungsvorlagen für eine kontextunabhängige Formalisierung

Das Beispiel zeigt eine PINCH-Geste, die durch das Spreizen der Finger auf einem Objekt zu beschreiben ist. Eine mögliche Folgeaktion ist das Skalieren dieses Objektes. Ein Beispiel für die Anwendung dieses Ansatzes ist, Standardgesten vorzudefinieren, um sie anschließend mit einem einfachen Funktionsaufruf an ein Objekt zu binden und so den Kontext zu bestimmen.

4.2.9 Anomalien

In diesem Abschnitt werden Ausprägungen der Beschreibungssprache diskutiert, die auf Basis der GeForMT-Grammatik syntaktisch zulässig, jedoch in der Implementierung uneindeutig oder nicht anwendbar sind.

In Online-Gesten sind komplexe Bewegungsformen schwer zu erfassen, da das Ereignisprofil der Eingabe zunächst unvollständig ist. Beispielsweise liegen beim Zeichnen einer Pfeilspitze, beschrieben durch den Ausdruck `LINE_SE`, `LINE_SW` erst ab dem Pfadknick ausreichend Daten vor, um die Geste zu klassifizieren. Die sequentielle Komposition ist demzufolge nicht sinnvoll. Aber auch die Erkennung einer atomaren Geste wie eine Kreisbewegung zur Eingabezeit ist in dieser Hinsicht eine Herausforderung. Die Formalisierung von Online-Gesten konzentriert sich damit auf eindimensionale und willkürliche Bewegungen.

*Komplexe
Bewegungsformen in
Online-Gesten*

Die Komposition zu Mischenformen aus zusammengesetzten Gesten und Multitouch-Gesten, setzt in ihrer Implementierung eine Priorisierung der Operationen voraus. An dieser Stelle wird eine in ihrer Implementierung nicht zu unterschätzende Komplexität zugelassen. Die Anwendbarkeit dieser Gesten ist abzuwägen und in weiterführenden Arbeiten anhand von Beispielen zu evaluieren.

*Komposition zu
Mischformen*

Relationen wie `SYNC`, `JOIN`, `SPLIT` wurden eingeführt, um räumliche Beziehungen zwischen ein oder mehreren synchron ausgeführten Bewegungen näher zu spezifizieren. Sprachbausteine wie `CROSS` und `CONNECT` konzentrieren sich hingegen auf sequentielle Kompositionen atomarer Gesten. In Kapitel 4.2.4 wurden weitere Vorschläge gemacht, die sich speziell auf die Anwendung in zusammengesetzten Gesten beziehen.

*Operationsbezogene
Relationen*

Die Spezialfälle zeigen den Bedarf einer Validierungsvorschrift der Sprache auf, die von der Grammatik nicht abgedeckt werden kann. Es ist notwendig die Sprachbausteine in ihrer Zusammenstellung zu analysieren und hinsichtlich ihrer Deutbarkeit und Anwendbarkeit zu bewerten.

4.2.10 Zusammenfassung

Das Kapitel schließt mit einer zusammenfassenden Darstellung der aufgezeigten Problemfelder ab. Diskutiert wurde der Sprachansatz vorrangig im Kontext einhergehender Web-Technologien, insbesondere in Bezug auf die Realisierung von Objekt- und Aktionsbeschreibung. Als weiterer thematischer Schwerpunkt, der das Sprachkonzept allgemein betrifft, wurde die sequentielle Komposition von Gesten behandelt. In Tabelle 4.2 werden die vorgestellten Optimierungsansätze und Erweiterungsvorschläge in einer Übersicht zusammengefasst.

Problemfeld	Ansätze der Optimierung
Sequentielle Komposition mit Komma-Operator ist uneindeutig definiert	Erweiterung um Semikolon-Operator lässt Unterscheidung zwischen Einzelpfad-Gesten und zusammengesetzten Gesten zu
Fehlen von Ansätzen einer relativen Lagebeschreibung in zusammengesetzten Gesten	Einführung von Relationen für Anknüpfung, horizontale und vertikale Nachbarschaft, sowie richtungsbasierte Nachbarschaft
Geschlossenheit einer Bewegungsform ausdrücken	Einführung der Relation CLOSE
Formalisierung des Objektbezugs im Kontext von Web-Anwendungen	Verwendung von Selektoren als Schnittstelle zu Interface-Elementen Spezifizierung des Objektkontext durch Interaktionsbereiche Erkennung von Gesten im Abseits von Objekten durch ereignisorientierten Ansatz mit den Sprachzusätzen ENTER und LEAVE oder durch Erweiterung des Erkennungsrahmens
Schnittstellendefinition für die Implementierung der Geste	Aktionszuordnung als Sprachzusatz Aktionszuordnung durch erweiterbare JavaScript-Datenstruktur (Konfigurations-Objekt)
Adaption der Wiederverwendung von Gestenbeschreibungen	Konkretisierung der Einschränkungen bei der Wiederverwendung von Gestenbezeichnern und Einführung von Trennzeichen Vorschlag der kontextunabhängigen Vordefinition von Gesten für die Verwendung als Vorlage

Tabelle 4.2: Übersicht der Optimierungs- und Erweiterungsvorschläge

Die Erweiterung von GeForMT um Ansätze zur sequentiellen Komposition ist besonders zu empfehlen, da sie Mehrdeutigkeiten bezüglich der Zusammensetzung

und Anordnung von Offline-Gesten auf ein Minimum einschränken. Die Konzeption der Schnittstellen zwischen Formalisierung und Implementierung ist vorrangig anwendungsspezifisch und bei einer Übernahme hinsichtlich der in Kapitel 4.2.7 beschriebenen Interoperabilität zu prüfen. Die Wiederverwendung ist ein Zusatz, der die Definition einer Vielzahl von Gesten erleichtern kann.

5 KONZEPTION UND IMPLEMENTIERUNG

Inhalt dieses Kapitels ist die softwaretechnische Realisierung des Web-Frameworks auf Basis der webkonformen Gestenformalisierung, die in Kapitel 4.2 erarbeitet wurde. In Kapitel 5.1 werden zunächst hauptsächlich nicht-funktionale Anforderungen an das Framework abgesteckt und deren Umsetzung in Hinblick auf die Abgrenzung der Kriterien erläutert. Im darauf folgenden Kapitel 5.2 werden die implementierten Komponenten und deren Entwurf erklärt. Die Darlegung der Funktionsweise des Erkennungsalgorithmus ist Schwerpunkt dieses Kapitels. (siehe 5.2.6) Anschließend wird die Installation (Abschnitt 5.3.1) und Anwendung (Abschnitt 5.3.2) des Frameworks dargelegt. In den letzten Abschnitten wird der praktische Bezug anhand von Anwendungsbeispielen hergestellt. Es wird zum einen die Testumgebung vorgestellt, in der verschiedenartige Gesten exemplarisch implementiert sind. (Abschnitt 5.3.3) Zum anderen wird der Prototyp einer mobilen Web-Seite präsentiert, der auf den konzeptionellen Überlegungen in Kapitel 4.1 aufbaut.

5.1 ANFORDERUNGSKRITERIEN

Ein wichtiges Kriterium für die sichere Anwendbarkeit eines Frameworks ist dessen Verständlichkeit. Unter Transparenz ist in dieser Hinsicht eine nachvollziehbare Arbeitsweise des Frameworks zu verstehen, ohne den Anwender mit Details der Verarbeitung zu überfordern. Dennoch soll der Entwickler auf wichtige Entscheidungsfragen durch die Bereitstellung von Optionen Einfluss nehmen können (Konfigurierbarkeit). Die Schnittstellen-Implementierung (API) folgt klaren Regelungen und einer verständlichen Sprache zu Gunsten der Einarbeitung und Benutzung des Frameworks. Diese Ziele gehen mit der Beschreibungssprache GeForMT konform, auf der das Framework aufbaut.

Transparenz

In Hinblick auf die weiterführende Optimierung des Frameworks wird eine modulare Implementierung der Komponenten vorgeschlagen, um die Wiederverwendung und Austauschbarkeit dieser zu ermöglichen. Die Umsetzung erfolgt auf Basis bewährter Entwurfs- und Coding-Muster wie beispielsweise das *Revealing Module Pattern*, das in Kapitel 2.3.5 vorgestellt wurde und unter anderem ein Konzept für die Strukturierung und Zugriffskontrolle des Codes bereitstellt.

Erweiterbarkeit

Kompatibilität stellt sich als Kriterium in zweierlei Ausprägungen dar. Zum einen den geräte- und plattformunabhängige Einsatz des Frameworks, zum anderen die Verwendung als Werkzeug neben anderen Bibliotheken, die sich beispielsweise auf

Kompatibilität

die grafische Gestaltung des User-Interface konzentrieren. Die Lösung der ersten Problematik stützt sich auf den Erkenntnissen in Kapitel 3.2. Anhaltspunkt für die plattformübergreifende Unterstützung des Frameworks sind die empfohlenen Standards des W3C. Darauf aufbauend ist eine Adaption von Maus- und Touch-Eingaben vorzunehmen, damit das Framework sowohl in desktopbasierten als auch mobilen Web-Browsern eingesetzt werden kann. Die Verknüpfung mit grafischen Bibliotheken wird in dieser Arbeit nicht betrachtet. Die Implementierung zielt jedoch auf unabhängige und erweiterbare Schnittstellen ab, um Entwicklern die Anbindung grafischer Komponenten und anwendungsspezifischer Funktionen zu ermöglichen.

Performanz

Die Verarbeitungsgeschwindigkeit bei der Gestenerkennung spielt eine wichtige Rolle in Hinblick auf die Akzeptanz gestenbasierter Bedienkonzepte. In Bezug auf die Framework-Implementierung wird dieser Aspekt vorallem bei der Wahl der Erkennungsmethode, die auf der Bewertung bestehender Ansätze in Kapitel 3.3.3.1 basiert, berücksichtigt. Des Weiteren wird auf dem Prototyp-Konzept von JavaScript (vgl. Kapitel 2.3.5) aufgebaut, sowie die Kompremierung des Codes in Betracht gezogen, um den Speicherbedarf und die Ladezeiten zu verringern.

Feedback

Die Rückmeldung wird in Kapitel 2.2.2 als elementare Eigenschaft einer Geste definiert und auch in Kapitel 3.4 wird dessen Stellenwert deutlich. Daher wird ein visuelles Feedback auf elementarer Ebene berücksichtigt und prototypisch als Teil des Frameworks umgesetzt. (siehe Kapitel 5.2.7) Prinzipiell soll dem Web-Entwickler jedoch freigestellt werden in welcher Art und Weise eine Rückmeldung erfolgt, insbesondere wenn es darum geht anwendungsspezifische Aktionen auszulösen. Das Rückmeldeverhalten wird üblicherweise auf Basis einer eventbasierten Architektur realisiert.

5.2 KOMPONENTEN DES FRAMEWORKS

Übersicht implementierter Module

Abbildung 5.1 gibt eine Übersicht über die Bestandteile des Frameworks, die jeweils als Module (vgl. Listing 2.5) realisiert sind. Frameworkinterne Module sind im Namensraum **GeForMT** zusammengefasst. Der Namensraum bildet gleichzeitig das Hauptmodul des Frameworks und stellt damit die Anwendungsschnittstelle für den Webprogrammierer bereit. Die Konzeption und Verwendung der API wird in Abschnitt 5.3.2 näher erläutert. Desweiteren regelt das Hauptmodul die Initialisierung und Verknüpfung aller Komponenten und steuert den Datenfluss.

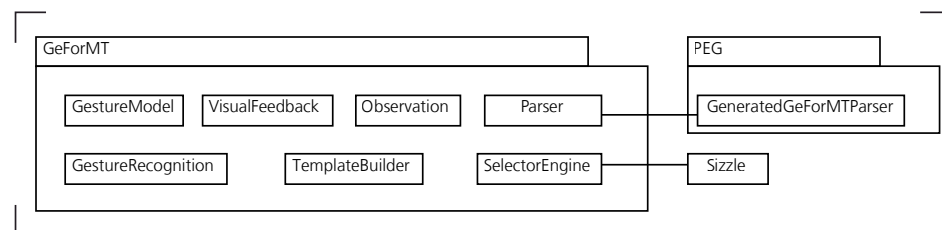


Abbildung 5.1: Module im Namensraum GeForMT und verwendete Bibliotheken

Verarbeitungsschritte einer Gestendefinition

Mit der Definition einer Geste auf Basis einer formalen Beschreibung werden im Wesentlichen sechs Verarbeitungsschritte durchlaufen. Zunächst wird der GeForMT-Ausdruck durch die **Parser**-Komponente validiert und in seine Syntaxbausteine zerlegt. (Abschnitt 5.2.2) Das **SelectorEngine**-Modul prüft enthaltene Selektoren und gibt die zugehörigen **Element**-Knoten des DOM-Baumes aus. (Abschnitt 5.2.3) Darauf aufbauend registriert sich das **Observation**-Modul für Maus- und Touch-Events

auf diesen Elementen. (Abschnitt 5.2.4) Im **TemplateBuilder**-Modul werden die formalen Parameter des geparsen Ausdrucks in eine mathematisch berechenbare Datenstruktur transformiert. (Abschnitt 5.2.6) Diese Vorlagen bilden die Vergleichsbasis für die Gestenerkennung, welche im Modul **GestureRecognition** vorgenommen wird. (Abschnitt 5.2.6) Zuvor werden die Ergebnisse der parsenden und transformierenden Komponenten in einem Objektmodell zusammengefasst und an zentraler Stelle im Modul **GestureModel** abgelegt. (Abschnitt 5.2.5)

Die folgenden Abschnitte behandeln den Aufbau und die Arbeitsweise der Module im Detail. Im Sinne einer Abbildung des Modul-Entwurfsmusters sind deren öffentliche Schnittstellen als eigenständige Objekte in den gezeigten UML-Diagrammen modelliert. Hinzufügend ist zu bemerken, dass die folgende Erläuterung nur die wichtigsten Operationen und Eigenschaften behandelt. Es existieren darüber hinaus weitere Hilfsfunktionen, deren Unterschlagung jedoch das Verständnis für die Konzeption des Frameworks nicht beeinträchtigt. Eine vollständige Darstellung der Funktionen ist der Dokumentation zu entnehmen.

Anmerkung

5.2.1 Globale Konstanten

Das Framework baut unmittelbar auf der Gestenbeschreibungssprache GeForMT auf. Insbesondere die Verwendung von Terminalsymbolen der Grammatik werden von verschiedenen Modulen geprüft, um das davon abhängige einzusetzende Verfahren zur Verarbeitung der Geste zu bestimmen. Zu Gunsten der Interoperabilität der Module, wird eine gemeinsame Verständigungsbasis für derartige Syntaxbausteine in Form globaler Konstanten geschaffen. Gemäß der Grammatikregeln werden die Konstanten klassifiziert. Daraus ergeben sich folgende Datentypen im GeForMT-Namensraum: **OPERATION_TYPES**, **RELATION_TYPES**, **FUNCTION_TYPES**, **CONTACT_TYPES**, **VECTOR_TYPES**, **SHAPE_TYPES**, **DIRECTION_TYPES** und **ROTATION_TYPES**. Sie repräsentieren jeweils eine Aufzählung der in GeForMT definierten Terminalsymbole.

*Repräsentation von
Terminalsymbolen der
Grammatik*

5.2.2 Parser

Das Parsen der definierten Gesten wird auf Grundlage der GeForMT-Grammatik vorgenommen, die als EBNF vorliegt. (vgl. Kapitel 3.3.2.3) Die formalen Ausdrücke werden dabei in ihre Syntaxbausteine zerlegt und in ein für die Weiterverarbeitung geeignetes Format transformiert. Das **Parser**-Modul verfügt über Methoden, die konkrete Parser-Implementierungen adaptieren. (siehe Abb. 5.2) Diese können mit der Methode **registerParser** eingebunden werden, indem der Name und die zu adaptierende Funktion übergeben wird. Der Zugriff auf die Operation erfolgt dann über die zentrale Methode **parse**. Dieses Entwurfsmuster vereinfacht die Erweiterung oder den Austausch konkreter Parserimplementierungen.

*Generalisierte
Schnittstelle*

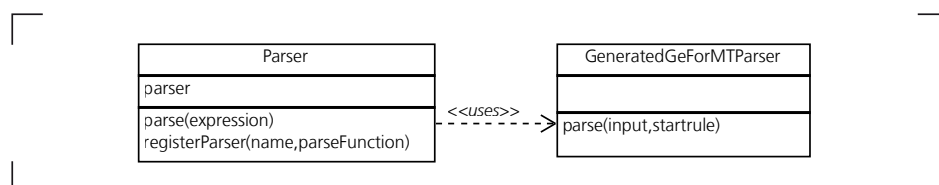


Abbildung 5.2: Adaption der Parserimplementierung

Für die Entwicklung einer konkreten Parser-Implementierung existieren generische Ansätze, die auf Basis der EBNF-Spezifikation entsprechenden Programmcode erzeugen - sogenannte Parsergeneratoren. Zur Generierung eines webkonformen Parsers wird die JavaScript-Bibliothek *PEG.js* eingesetzt (siehe [Majda, 2010]). Sie basiert auf dem *Parsing Expression Grammar*-Formalismus (PEG) - einem Grammatiktyp, der einen linearen, ordnungsabhängigen Analyseprozess vorsieht. *PEG.js* orientiert sich an der geläufigen EBNF-Syntax, erweitert diese darüber hinaus um JavaScript-ähnliche Sprachoptionen. Sie erlauben JavaScript-Code innerhalb der Grammatik-Spezifikation. Dies ermöglicht eine direkte Adaption der auszugegebene Datenstruktur. Ergebnis ist ein Objektmodell, in dem die in Abschnitt 5.2.1 beschriebenen global definierte Konstanten bereits angewendet werden. Listing 5.1 zeigt die entsprechende Grammatik am Beispiel der GeForMT-Regeln *definition* und *function*.

```
definition = complex:complex opt:(option complex)*
{
  var definition={options: []};
  definition.options.push(complex);
  for (var i=0;i<opt.length;i++){
    definition.options.push(opt[i][1]);
  }
  return definition;
}

function = num:integer?functionType:'F'
{
  var func = {type:GeForMT.FUNCTION_TYPES.FINGER,number:null};
  if(typeof num != 'undefined') func['number']=num;
  return func;
}
```

Listing 5.1: Auszug der Grammatik nach PEG.js-Spezifikation für die
Parser-Generierung

Die Generierung kann sowohl über Kommandozeilen-Befehle als auch über die JavaScript-API im Browser vorgenommen werden. Die Parsergenerator-API lässt sich theoretisch direkt in das Framework integrieren. Da die dynamische Parsergenerierung jedoch nur für die Frameworkentwicklung und nicht für den Einsatz in einer Webanwendung Bedeutung hat, wird der generierte Parser statisch unter dem Modulnamen **GeneratedGeForMTParser** im Namensraum PEG eingebunden und anschließend vom generalisierten Parser adaptiert.

5.2.3 Selector Engine

Das theoretische Konzept zur Herstellung des Objektbezugs einer Geste wurde in Kapitel 4.2.5 erläutert. Demnach wird der in GeForMT als **focus** definierte Geltungsbereich einer Geste durch Selektoren beschrieben, die eine Auswahl von HTML-Elementen treffen. In dem Modul **SelectorEngine** wird das selbe Entwurfsmuster für eine generalisierte Schnittstelle angewendet wie in der Parser-Komponente. (siehe Abb. 5.3) Konkrete Engine-Implementierungen werden demnach adaptiert und der Zugriff über diese Schnittstelle geregelt. Somit können auch hier externe Bibliotheken über die Methode **registerEngine** einfach eingebunden und ausgetauscht werden. Die Kernfunktionalität wird mit der Methode **getElements** bereitgestellt, die einen oder mehrere Selektor-Ausdrücke entgegennimmt und eine Liste von DOM-Objekten vom Typ **Element** zurückliefert.

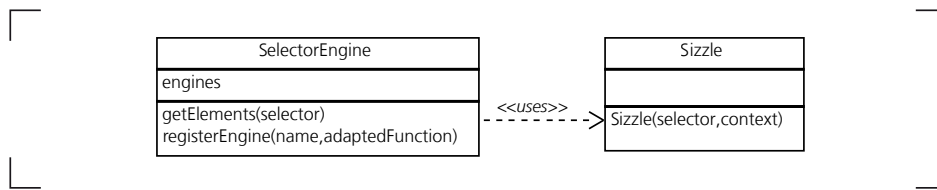


Abbildung 5.3: Adaption der Selector Engine Sizzle

Mit *Sizzle* [Sizzle, 2012] wird eine alleinstehende, browserübergreifende Lösung einer Selector-Engine angeboten, die auch im JavaScript-Framework *jQuery* (vgl. [jQuery, 2011]) verbaut ist. Das Parsen der Ausdrücke und die Selektion zugeordneter DOM-Objekte wird dadurch weitestgehend abgenommen. Die API-Methode *Sizzle* ist Einstiegspunkt für die Operation.

Sizzle

5.2.4 Ereignisverarbeitung

Die Entgegennahme und Verarbeitung elementarer Ereignisse der Nutzereingabe liegt im Zuständigkeitsbereich des **GeForMT.Observation**-Moduls. Im Sinne einer losen Kopplung wird das *Observer*-Muster in Anlehnung an [Stefanov, 2011, S. 174] umgesetzt, das es anderen Komponenten des Frameworks erlaubt sich für Nachrichten der Observation zu registrieren. Sie nehmen die Rolle des *Beobachters* ein und müssen entsprechende *Callback*-Methoden implementieren.

Observation-Modul

Das Modul verwaltet **EventObservation**-Objekte, die die Interaktion auf genau einem DOM-Element registriert und verarbeitet. Sie implementieren demnach Event-Handler sowohl für Touch- als auch für Maus-Ereignisse. In Kapitel 4.2.1 und 4.2.2 wurde die Beschreibbarkeit von Mausgesten mit GeForMT und deren Gemeinsamkeiten mit Touch-Eingaben bereits diskutiert. Eine **EventObservation** führt außerdem einen Katalog zugehöriger Gesten, der es erlaubt für jedes Ereignis eine Auswahl relevanter Gesten zu treffen.

Event-Registrierung

Das zu Grunde liegende Event-Interface orientiert sich an den in Kapitel 3.2.1 beschriebenen Konventionen für Touch-Events. Um auch Maus-Ereignisse gleichermaßen verarbeiten zu können, wird eine Transformation des Event-Objektes nach Touch-Event-Spezifikation vorgeschlagen. Angelehnt an den Ansatz in *Phantom Limb* (vgl. Kapitel 3.3.1) und *MagicTouch* (vgl. Kapitel 3.2.2) wird die Emulation durch Absetzung eines neuen Events mit der JavaScript-Methode **dispatchEvent** (vgl. Kapitel 2.3.6) realisiert. Dieses wird dann von den touchspezifischen Event-Handleern empfangen und damit Single-Touch-Gesten simuliert.

Vereinheitlichung des Event-Interface

Die Prinzipien der Event-Ausbreitung wurden in Kapitel 2.3.6 behandelt. Als Strategie für die Ereignisbehandlung wird das *Bubbling* gewählt. Diese Herangehensweise liegt in dem folgenden Fallbeispiel begründet: Angenommen der Web-Seitenentwickler definiert sowohl eine freie Geste (gebunden an das Element **html**), als auch ein objektorientierte Geste auf einem kleineren **div**-Element. Der User beabsichtigt nun eine freie Geste auszuführen, durchquert dabei jedoch das enthaltene **div**-Element oder beginnt die Geste sogar darin. Die Ereignisse, die auf dem **div** registriert wurden gehören potentiell auch zu der freien Gesten auf dem Eltern-Knoten und müssen demnach auch als solche registriert werden. Durch das *Bubbling* wird das Event an die Elternelemente vererbt und so eine Blockade vermieden.

Vererbung von Events durch Bubbling

Das Browser-Plugin *npTUIOClient* und die JavaScript-Connector-Implementierung *MagicTouch*, die in Kapitel 3.2.2 behandelt wurden, sind eine optimale Erweiterung des Event-Systems, um auch TUIO-basierte Systeme zu unterstützen. Die Lösungen sind vom Framework unabhängig und können bei Bedarf zusätzlich installiert werden. Insbesondere für das Testen des Frameworks auf dem Entwicklungssystem ist ihr Einsatz von Vorteil. Damit ist es beispielsweise möglich, Touch-Eingaben mit dem Simulator des *ReactIVision*-Frameworks [Kaltenbrunner und Bencina, 2009] zu simulieren. Noch komfortabler ist das Testen unter der Verwendung eines Multitouch-Gerätes wie zum Beispiel ein Tablet, Trackpad oder Touchscreen-Monitor. Mögliche Tracker- und Serverimplementierungen, die hierfür verwendet werden können, sind in [Kaltenbrunner, 2011] zusammengefasst.

5.2.5 Datenmodell

Das **GestureModel**-Modul fasst alle Ergebnisse des Verarbeitungsprozesses einer GeForMT-Definition zusammen und speichert sie für die Laufzeit der Anwendung. Es bildet damit das Kernstück des Frameworks, aus dem der Gestenerkennner letztlich Informationen über die Parameter formal definierter Gesten beziehen kann. Dies erfordert eine vereinfachte Repräsentation einer GeForMT-Geste in Form eines Objektmodells, das in Abbildung 5.4 veranschaulicht wird.

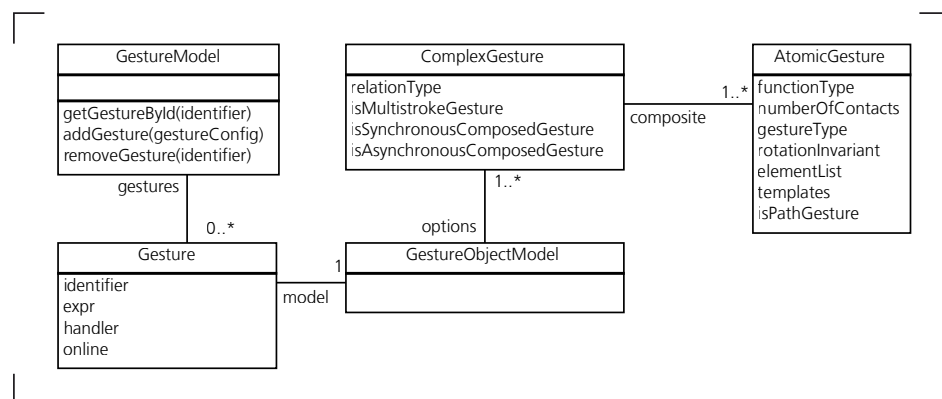


Abbildung 5.4: Zentrales Datenmodell

Eine Geste wird durch die Klasse **Gesture** abgebildet. Es enthält in erster Linie die Parameter, die bei der Definition in der Hauptschnittstelle angegeben wurden. (vgl. Kapitel 5.3.2) Die Abbildung des GeForMT-Ausdruckes als Objektmodell wurde so weit vereinfacht, dass die Geste in drei Klassen modelliert werden kann. Sie repräsentieren im weitesten Sinne die Grammatik-Regeln *definition*, *complex* und *gesture*. (vgl. Kapitel 3.3.2.3) **GestureObjectModel** hält folglich eine Liste von **ComplexGesture**-Objekten als Optionen. Dieses Konstrukt wurde in [Henzen, 2010] mit dem Operator 'I' eingeführt, um Variationen ausdrücken zu können. **ComplexGesture** bildet eine Komposition mehrerer atomarer Gesten ab. Als Eigenschaften werden die Art der Komposition und die Definition einer Relation festgehalten. Die Klasse **AtomicGesture** ist als kleinste Einheit einer Gestenbeschreibung zu verstehen. Die Kontaktart (**functionType**), die Anzahl der Kontakte (**numberOfContacts**), die Bewegungsart (**gestureType**) und möglichen Geltungsbereichen (**elementList**) der Geste fassen die Eigenschaften einer atomaren Geste zusammen, die sich aus dem geparsten GeForMT-Ausdruck extrahieren lassen. Darüberhinaus wird aus der Angabe von Richtungsangaben interpretiert, ob in dieser Hinsicht Variationen zugelassen sind (**rotationInvariant**). Ebenso wird aus der Art der Geste geschlossen, ob es

sich um eine pfadbasierte Geste handelt (`isPathGesture`). Eine Besonderheit stellt die Eigenschaft `templates` dar. Sie enthält eine oder mehrere `Template`-Objekte, die in Hinblick auf eine mathematische Berechnung in der Gestenerkennung eine geeignetere Repräsentation der Geste darstellen. Näheres zu dieser Datenstruktur wird in Abschnitt 5.2.6 behandelt. Das `GestureObjectModel`-Modul stellt selbst keine Funktionalität für die semantische Auswertung der Bausteine eines GeForMT-Ausdruckes bereit, sondern unterstützt lediglich die Datenhaltung. Eine Adaption der Daten, die vom Parser geliefert wird und die Überführung in das erläuterte Objektmodell wird im Hauptmodell vorgenommen.

5.2.6 Template-Erzeugung und Gestenerkennung

In Kapitel 3.3.3 wurden verschiedene Verfahren der Mustererkennung vorgestellt und für den Einsatz in einem sprachbasierten Web-Framework bewertet. Template-Matching-Verfahren wie der \$1-Recognizer sind aufgrund ihres ausgewogenen Verhältnisses von Erkennungsrate, Speicherbedarf, Toleranz und Umsetzbarkeit in einer JavaScript-Implementierung eine geeignete Wahl. Ein entscheidender Vorteil des Verfahrens ist der geringe Aufwand hinsichtlich der Merkmalsextraktion, die während der Laufzeit der Anwendung erfolgt. Die zu Grunde liegende Datenstruktur besteht aus normierten Koordinaten, die einen Gestenpfad repräsentieren. An dieser Stelle setzt das Modul `TemplateBuilder` an.

Template-Matching als Erkennungsverfahren

Das Modul erzeugt sowohl aus der formalen Gestenbeschreibung, als auch auf der Basis von Ereignissen `Template`-Objekte, eine Punktliste für jeden Gestenpfad. Jedes Template erhält einen Identifikator (`identifier`), der es insbesondere bei Multitouch-Gesten erlaubt, einzelne Berührungspunkte in Beziehung zu setzen und damit einem Pfad zuzuordnen. Abbildung 5.5 veranschaulicht die Modellierung.

TemplateBuilder-Modul

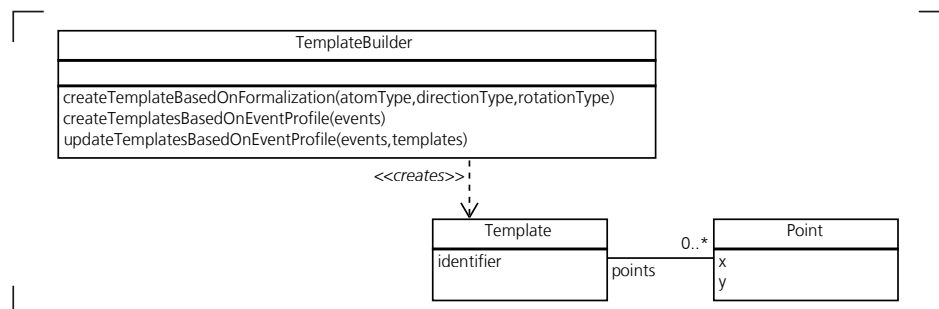


Abbildung 5.5: TemplateBuilder-Modul

Das Schema in Abbildung 5.6 verdeutlicht die Zusammenhänge der drei Hauptaspekte der Gestenerkennung (vgl. Kapitel 3.3.3.1) am Beispiel einer L-Geste. Ausgehend von den in GeForMT beschriebenen Eigenschaften einer atomaren Geste, wie die Art der Bewegung, Bewegungsrichtung und Rotation, werden Punkte erzeugt, die den idealen Bewegungspfad beschreiben. Dabei sind Anzahl und Abstände der Punkte einheitlich. Im Fall einer Komposition von Gestenatomen mit dem Komma-Operator, wie es im Beispiel gezeigt wird, werden die Pfade konkateniert. Resultat ist ein auf Koordinaten basiertes Referenzmuster. Die Eingabe des Nutzers besteht zunächst aus einer Folge von Ereignissen, aus denen lediglich die tatsächlichen Koordinaten zu entnehmen sind. Eine Ähnlichkeit zwischen Referenzmuster und ausgeführter Geste kann nur unter den gleichen Bedingungen festgestellt werden, weshalb die Punktpfade im nächsten Schritt normiert werden, wie es in Kapitel 3.3.3

Erkennung von GeForMT-Gesten mit \$1

beschrieben wurde. Danach kann die durchschnittliche Distanz zwischen den Punkt-Mustern ermittelt werden. Liegen mehrere Gestenbeschreibungen und damit mehrere Referenzmuster vor, sind dementsprechend viele Berechnungen vorzunehmen. Die Geste mit der geringsten Abweichung gilt als erkannt. Der Algorithmus wird um einen Grenzwert der maximalen Abweichung erweitert, damit Fehleingaben nicht als Systemanweisung interpretiert werden.

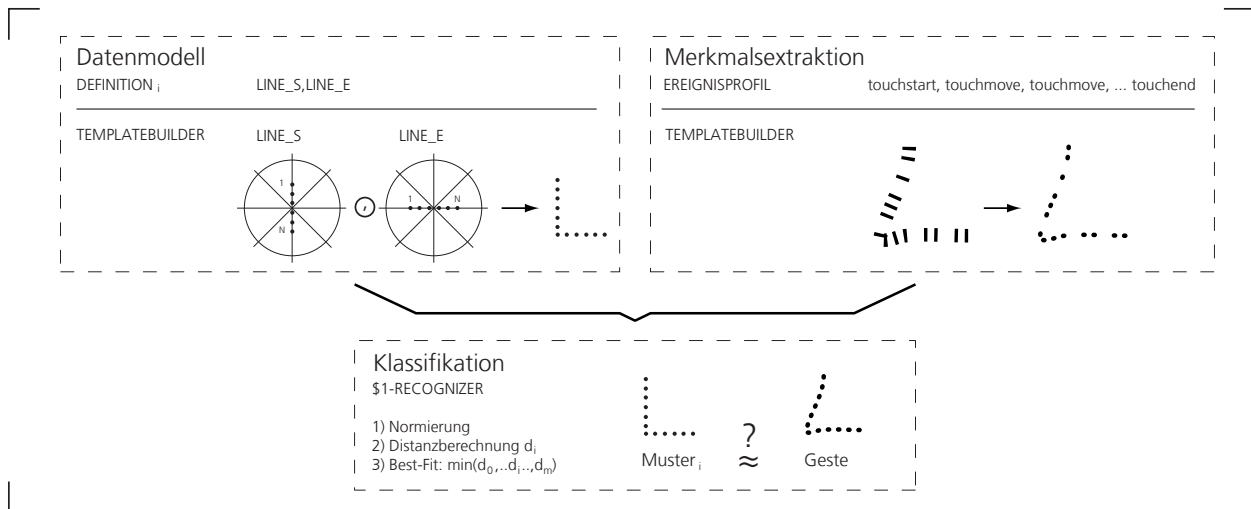


Abbildung 5.6: Erkennung pfadbasierter Gesten basierend auf GeForMT und dem Verfahren nach [Wobbrock et al., 2007]

*Adaption des
Algorithmus bezüglich
der Varianz*

Das vorgestellte Verfahren wirft die Frage auf, wie mit der Formalisierung ohne Angabe einer Bewegungsrichtung zu verfahren ist. Eine Lösung hierfür ist bereits im \$-Algorithmus integriert. Die Erkennung von Varianten wird erhöht, indem die Templates einheitlich auf 0° gedreht werden. Das Framework baut darauf auf und nutzt dieses Vorgehen bei unkonkreten Gestenbeschreibungen aus. Es kann jedoch nicht auf die Rotationsrichtung bei kreisförmige Gesten angewendet werden. Ebenso sind komponierte Einzelpfad-Gesten davon ausgenommen (Beispiel: $1F(LINE, LINE)$). In den Fällen werden variierende Referenzmuster konstruiert, die jeweils mit der Eingabe verglichen werden müssen.

*Erkennung
zusammengesetzter
Gesten*

Das dargelegte Beispiel in Abb. 5.6 erklärt das Verfahren bei Einzelpfad-Gesten. Die Verarbeitung zusammengesetzter Gesten (vgl. Kapitel 4.2.4) liegt nah: Bei zwei sequentiell ausgeführten Gestenpfaden, wird zunächst der erste Teil ausgewertet und wenn dieser erkannt ist der Zweite. Bedeutend ist hierfür die Implementierung eines Zeitfensters (als **contiguityIntervalTimer**, vgl. Abb. 5.7) - dem in Kapitel 2.2.3 definierten Kontiguitätsintervall. Innerhalb dieses Intervalls werden weitere gestische Eingaben erwartet, die jeweils geprüft werden bis alle atomaren Bestandteile der Definition abgearbeitet sind. Der erweiterte Erkennungsalgorithmus entspricht in den Grundzügen dem \$N-Recognizer von [Anthony und Wobbrock, 2010] (vgl. 3.3.3), unterscheidet sich jedoch in den Punkten des Auswertungszeitpunktes und den Mechanismen, um Variationen abzudecken.

*Erkennung von
Multitouch-Gesten*

Die Erkennung zeitgleich ausgeführter Eingaben, wie sie nach GeForMT-Spezifikation durch die synchrone und asynchrone Komposition und die Anzahl von Kontakten vorgesehen ist, erfordert ebenso Erweiterungen des Algorithmus. Die Verfahrensweise, wie sie in Abbildung 5.6 dargestellt ist, ändert sich im Wesentlichen nicht. Es erhöht sich allerdings die Anzahl der Templates, die abzugleichen sind. Das folgende Beispiel veranschaulicht dies. Definiert wird eine Geste, die als zwei auseinanderfliehende Linien beschrieben werden kann: $1F(LINE\_W)*1F(LINE\_E)$.

Im Datenmodell werden hierfür zwei Referenzmuster abgelegt. Führt der Nutzer nun eine Bewegung mit zwei Fingern aus, besteht die Herausforderung ihrer Interpretation darin zu erkennen, welches der beiden Referenzmuster zu welchem der beiden Bewegungspfade gehört und ob diese eine Ähnlichkeit aufweisen. Zu lösen ist demnach ein Korrespondenzproblem. In diesem Fall ist der Verarbeitungsaufwand mit insgesamt vier Berechnungsschritten vertretbar. Er erhöht sich jedoch mit der Zunahme simultaner Kontakte und weiteren ähnlichen Kandidaten. Dies ist beim Interaktionsdesign mit GeForMT zu berücksichtigen.

Die Erkennung von Gesten zur Eingabezeit wird ebenfalls mit dem zuvor beschriebenen Verfahren realisiert. In diesem Fall wird der Algorithmus mehrfach durchlaufen, während das Referenzmuster mit jeder Bewegung erweitert wird. Demnach wird immer der gesamte Bewegungspfad in der Berechnung berücksichtigt. Die Grenzen der Online-Gestenerkennung wurden in Kapitel 4.2.9 dargelegt und sind in der Formalisierung zu berücksichtigen.

*Erkennung von
Online-Gesten*

Abbildung 5.7 zeigt den Entwurf des Moduls **GestureRecognition**. Es ist als Beobachter des **Observation**-Moduls registriert und implementiert demnach Eventhandler, die die adaptierten Nutzereingaben in Empfang nehmen.

Entwurf des Gestenerkennungsmoduls

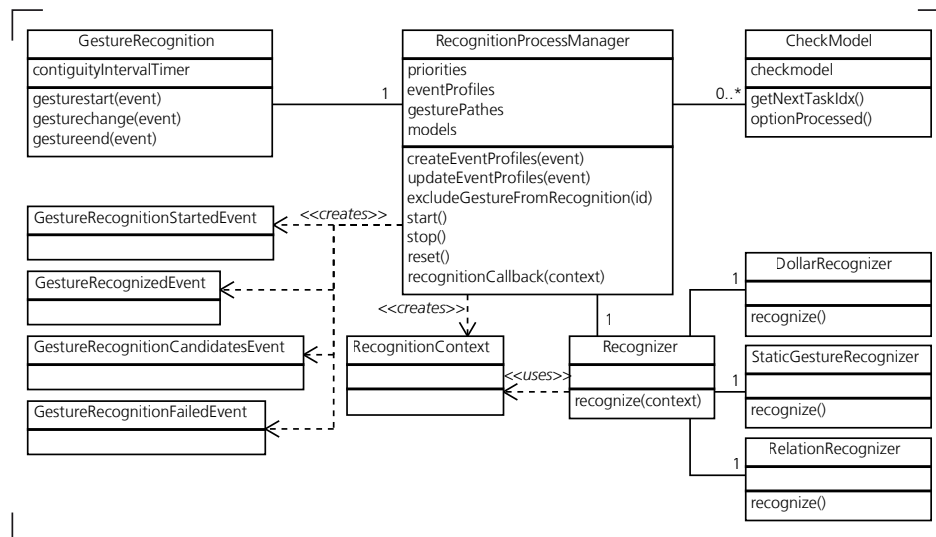


Abbildung 5.7: GestureRecognition-Modul

Der zentral gelegene **RecognitionProcessManager** steuert die wichtigsten Prozesse und entscheidet wann eine Geste beginnt, endet, zu interpretieren ist und als erkannt gilt. Es stehen ihm vier Ereignistypen zur Verfügung, die den Status der Gestenerkennung vermitteln:

- **GestureRecognitionStartedEvent**: Tritt ein, wenn die Berechnung startet.
- **GestureRecognizedEvent**: Tritt ein sobald eine Geste als erkannt gilt und liefert hauptsächlich das zusammengestellte Ereignisprofil und die daraus extrahierten Punktpfade für die Weiterverarbeitung aus.
- **GestureRecognitionCandidatesEvent**: Vermittelt Zwischenergebnisse, wenn zusammengesetzte Gesten verarbeitet werden. Es tritt ein sobald ein Teil der Geste erkannt wurde und listet potenzielle Kandidaten auf.
- **GestureRecognitionFailedEvent**: Tritt ein, wenn der Gestenerkenner keine Ergebnisse hervorbringt, was bedeutet, dass die Eingabe auf keine Beschreibung passt.

<i>Datenmanagement</i>	Des Weiteren stellt der RecognitionProcessManager eingehende Touch-Events und deren enthaltene Daten in einem Profil zusammen. Unter priorities werden potenzielle Kandidaten aufgelistet. Für jeden Kandidaten werden das Eventprofil (eventProfiles) und Templates unter Zuhilfenahme des TemplateBuilder -Moduls (gesturePathes) aktualisiert. Außerdem wird das jeweilige Objektmodell (models) bereitgestellt. Daran lehnt die Klasse CheckModel an. Sie bildet die Bestandteile einer Geste als eine Art Aufgaben-Checkliste für den Erkenner ab. Somit kann geprüft werden, ob eine Geste vollständig erkannt ist und wenn nicht, welcher Bestandteil als nächstes zu prüfen ist.
<i>Anwendung von Methoden der Gestenerkennung</i>	Der Recognizer ist als generalisierte Schnittstelle für die Erkennungsroutinen implementiert. Alle Anfragen zur Auswertung einer Geste werden an dessen Methode recognize gerichtet. Der RecognitionProcessManager schnürt hierfür ein Informationspaket, das im Diagramm in Abbildung 5.7 als RecognitionContext zu finden ist und der besagten Methode übergeben wird. Es enthält alle bis zum Zeitpunkt der Anfrage gesammelten Daten (Kandidaten, Eventprofile, Templates, Objektmodelle, etc.). Je nach Kontext und Problemstellung wird darauf aufbauend ein konkretes Erkennungsverfahren angewandt. Das bedeutenste Verfahren, der \$1-Recognizer , wurde in den vorangegangenen Absätzen bereits erläutert und ist im DollarRecognizer implementiert. Da diese Methode nur auf pfadbasierte Gesten anwendbar ist und sich auf die Erkennung der Bewegungsform konzentriert, werden erweiternde Implementierungen angeboten. Der StaticGestureRecognizer deckt beispielsweise die Erkennung punkthafter Gesten ab. Der RelationRecognizer prüft geforderte Eigenschaften gemäß der in GeForMT spezifizierten Relationen.
<i>Erweiterung um andere Template-Matching-Verfahren</i>	Die Konzeption des Gestenerkennungsmoduls erlaubt es, in wenigen Schritten die Erkennungsmethode zu ersetzen oder zu erweitern. So könnte in weiterführenden Arbeiten beispielsweise die Online-Gestenerkennung als spezialisierte Implementierung neben DollarRecognizer realisiert werden. Zum Beispiel könnte das SiGer -Verfahren nach [Swigart, 2005] (vgl. Kapitel 3.3.3) als Alternative implementiert werden. In dem Fall würde das Template -Objekt um formale Richtungsangaben erweitert werden. Die Auswertung dieser erfolgt dann auf Basis einer Anpassung der Hauptschnittstelle im Recognizer , in der SiGer eingebunden wird.

5.2.7 Feedback

<i>Schnittstelle des Moduls</i>	Das Modul VisualFeedback deckt die Anforderung an ein elementares visuelles Feedback ab. Mit der öffentlichen Methode setVisualFeedback , die einen boolschen Wert als Parameter nimmt, kann die Option aktiviert oder unterbunden werden. Der Webentwickler kann diese Einstellung über die Hauptschnittstelle in der Konfiguration vornehmen. (siehe Kapitel 5.3.2)
<i>Grafischer Kontext</i>	Ein grafischer Kontext wird durch das canvas -Element bereitgestellt, das mit HTML 5 eingeführt wurde. JavaScript stellt hierfür Zeichenfunktionen bereit, die es erlauben zweidimensionale Grafiken dynamisch zu erzeugen und auf der Zeichenfläche abzubilden. Die Verwendung als Feedbackkomponente erfordert, dass das canvas eine über allen Webinhalten übergeordnete transparente Schicht bildet und Eigenschaften bezüglich der Gestaltung und Interaktion nicht beeinflusst.
<i>Implementierung</i>	Bei der Initialisierung des Moduls werden zwei canvas -Elemente mittels JavaScript-Methoden zur DOM-Manipulation erzeugt und bei Aktivierung des Feedbacks dem body des HTML-Dokumentes hinzugefügt. Das erste canvas wird dafür genutzt, um

ein kontinuierliches Feedback über die aktuelle Position der Eingabe zu geben. Entsprechend wird jeder Kontakt als Kreis dargestellt. Auf dem zweiten **canvas** wird der Pfad der Geste und visualisiert. Grund der Verteilung des Feedbacks auf zwei Schichten ist einerseits die Trennung der Funktionalität (Kontakt- und Gestenpfad-visualisierung), andererseits die Berücksichtigung der Performanz bei der Aktualisierung des Gestenpfades. Ein hoher **z-index** sorgt dafür, dass die Zeichenflächen über anderen Webinhalten liegen. Ihre räumlichen Dimensionen richtet sich nach der Fenstergröße, so dass die visuelle Rückmeldung konsequent über die gesamte Benutzeroberfläche erfolgt. Abbildung 5.8 skizziert den Aufbau der Schichten.

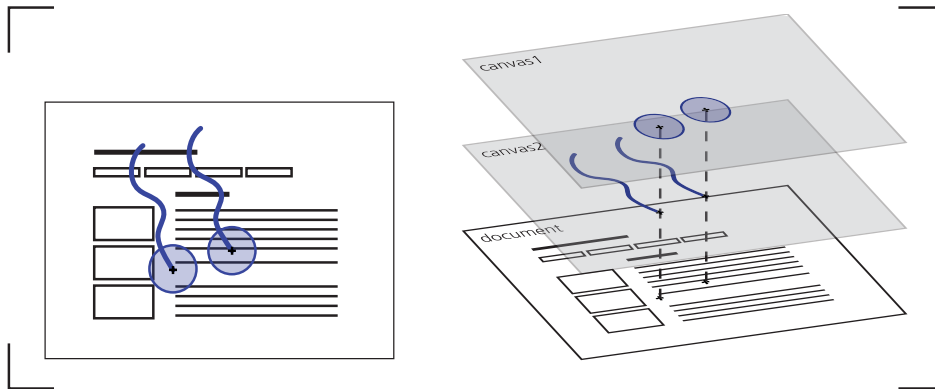


Abbildung 5.8: Zwei Zeichenflächen für die Visualisierung der Gestenpfade und aktuellen Kontaktpunkte über den Inhalten des Webdokumentes

Das Modul knüpft an die in Abschnitt 5.2.4 und 5.2.6 beschriebenen Module an, indem es sich für die Ereignisse der Module **Observation** und **GestureRecognition** registriert. Es fließen folglich sowohl aktuelle Positionsdaten, als auch der Status der Gestenerkennung in die Visualisierung ein.

Folgende Herausforderung ergibt sich aus dem beschriebenen Ansatz: In der Regel werden Ereignisse nur nach dem Prinzip des hierarchischen Ausbreitung vererbt. (siehe Kapitel 2.3.6 und Konzeption in Abschnitt 5.2.4) Die **canvas**-Elemente sind jedoch von dieser Hierarchie ausgeschlossen und blockieren den Eventfluss, wenn sie die Benutzeroberfläche überlagern. Das Problem wird mit der CSS-Eigenschaft **pointer-events** gelöst, die im aktuellen Arbeitsentwurf von CSS 3 definiert ist [W3C, 2011a] und nach [Deveria, 2011a] weitestgehend von den Browsern unterstützt wird. Mit dieser Eigenschaft werden die **canvas**-Elemente von der Ereignisverarbeitung ausgeklammert und somit durchlässig.

*Problematik der
Durchlässigkeit von
Ereignissen*

5.3 ANWENDUNG

5.3.1 Installation

*Einbindung in eine
Webanwendung*

Die Module, die das Framework bilden (vgl. Kapitel 5.2), sind in einer einzigen JavaScript-Datei zusammengefasst. Die Integration in eine Web-Seite erfolgt über das `script`-Tag in HTML, wie in Listing 5.2 gezeigt wird.

```
1  ..  
2  <head>  
3      <script type="text/javascript" src="GeForMT.min.js"></script>  
4  </head>  
5  ..
```

Listing 5.2: Einbindung des Frameworks in HTML

*Komprimierte und volle
Version*

Das Volumen der Datei wird durch die Komprimierung des Quelltextes (Verkürzung von Bezeichnern und Filterung von Kommentaren und Leerzeichen) reduziert, um die Ladezeit zu verringern. Als Endprodukt entsteht demnach eine minimale Version (vgl. Listing 5.2) und eine Vollversion (*GeForMT.full.js*), die Namenskonventionen beibehält und die Code-Dokumentation beinhaltet.

5.3.2 Anwendungsschnittstellen

Das Framework-API ist über den Namensraum **GeForMT** zugänglich. (vgl. Kapitel 5.2) Abbildung 5.9 zeigt die Funktionen, deren Anwendung im Folgenden erläutert werden.

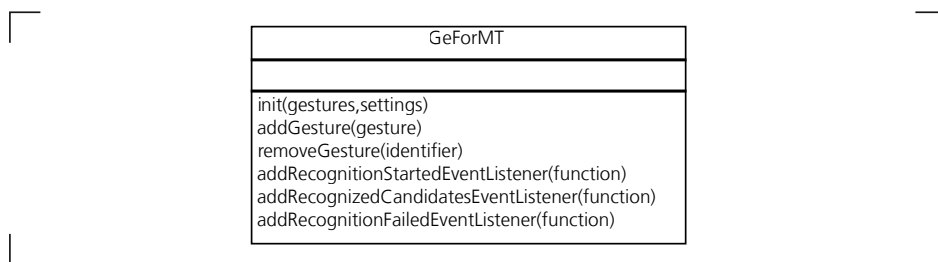


Abbildung 5.9: Haupt-Programmierschnittstelle

Initialisierung

Die Initialisierung der Framework-Komponenten durch die Methode `init` ist obligatorisch. Dabei können optional globale Einstellungen (**settings**) vorgenommen und eine Menge vordefinierter Gestenbeschreibungen (**gestures**) übernommen werden. Im einfachsten Fall ist die Verwendung dieser Funktion ausreichend, da hier bereits alle notwendigen Daten eingespeist werden können. Es ist zu empfehlen, die Initialisierung beim Laden des Webdokumentes durchzuführen, wie es in Listing 5.3 dokumentiert ist.

```

1 <script type="text/javascript">
2     window.onload = function(){
3         GeForMT.init(gestures, settings);
4     };
5 </script>

```

Listing 5.3: Initialisierung beim Laden des HTML-Dokumentes

Listing 5.4 zeigt die Anwendung einer Konfiguration und der Einstellungsparameter, die in der vorliegenden Version implementiert sind. Die Einstellung **feedback** entscheidet über die Visualisierung des Gestenpfades (vgl. Kapitel 5.2.7), durch **preventDefault** lässt sich das Standardverhalten des Browsers unterdrücken. Letzte Eigenschaft legt das Kontiguitätsintervall in Millisekunden fest. (vgl. Kapitel 5.2.6) Die Konfiguration ist optional und greift standardmäßig auf eine Voreinstellung zurück.

Globale Einstellungen

```

1 var settings = {
2     feedback: true,    // default true
3     preventDefault: false, // default true
4     contiguityInterval: 800 // default 1000
5 };

```

Listing 5.4: Einbindung des Frameworks in HTML

Kern einer Gestendefinition ist die Gestenbeschreibung in GeForMT. In den Kapiteln 4.2.6 und 4.2.7 wurden Möglichkeiten diskutiert, wie Formalisierung, Aktion und Konfiguration in Beziehung gesetzt werden können. Bezüglich der Erweiterbarkeit und Wiederverwendung überwiegen die Vorteile bei der Realisierung als Konfigurations-Objekt. Listing 5.5 (Zeile 1-9) konkretisiert den Ansatz am Beispiel. Ein Gesten-Set, wie es bei der Initialisierung übergeben werden kann, besteht folglich aus mehreren solcher Objekt-Literale, die in einem Sammel-Objekt zusammengefasst sind.

Definition von Gesten

JavaScript verleiht Web-Seiten aus technischer Sicht eine gewisse Dynamik in Hinblick auf deren Inhalte und Aussehen. Dieser Aspekt wird auch für die Interaktion berücksichtigt, indem Methoden zum Hinzufügen (**addGesture**) und Entfernen (**removeGesture**) von Gestendefinitionen zur Laufzeit angeboten werden.

Anwendung zur Laufzeit

```

1 // Definition
2 var gesture = {
3     identifier: "movePic",
4     expr: "MOVE(img.pic)",
5     online: true,
6     handler: function(e){
7         ..
8     }
9 };
10 // Hinzufügen einer Geste
11 GeForMT.addGesture(gesture);
12 // Geste entfernen
13 GeForMT.removeGesture("movePic");

```

Listing 5.5: Definition einer Geste und Registrierung

Status des Erkennungsprozesses

Ergänzt werden die Hauptschnittstellen durch die Registrierung von Callback-Methoden, in denen das Verhalten bei eintretenden Ereignissen des Gestenerkennungs-Moduls bestimmt werden kann. (vgl. Abb. 5.9) Diese wurden in Kapitel 5.2.6 beschrieben. Der Webprogrammierer erhält dadurch die Möglichkeit auf Fehleingaben entsprechend zu reagieren und beispielsweise eine Hilfe anzubieten.

5.3.3 Testumgebung

Aufbau der Testanwendung

Ein Nebenprodukt der Framework-Entwicklung ist eine HTML-basierte Testumgebung, deren Benutzeroberfläche sich in drei Bereiche unterteilt. Anhand eines Screenshots in Abbildung 5.10 können diese nachvollzogen werden. Im linken Teil können Einstellungen vorgenommen werden, vordefinierte Gesten geladen werden, sowie eigene Gesten basierend auf GeForMT definiert werden. Der grau hinterlegte Bereich, sowie die enthaltenen blau hinterlegten Quadrate sind als `div`-Elemente implementiert und stellen die Testobjekte dar, auf denen Gesten angewendet werden können. Die entsprechenden Selektor-Ausdrücke werden für die Verwendung in der Definition benutzerdefinierter Gesten im jeweiligen Objekt angezeigt. Unter der Zeichenfläche ist eine Konsole integriert, die den Status der Gestenerkennung ausgibt. In der Konsole kann demnach verfolgt werden, ob eine Geste erkannt wurde oder nicht.

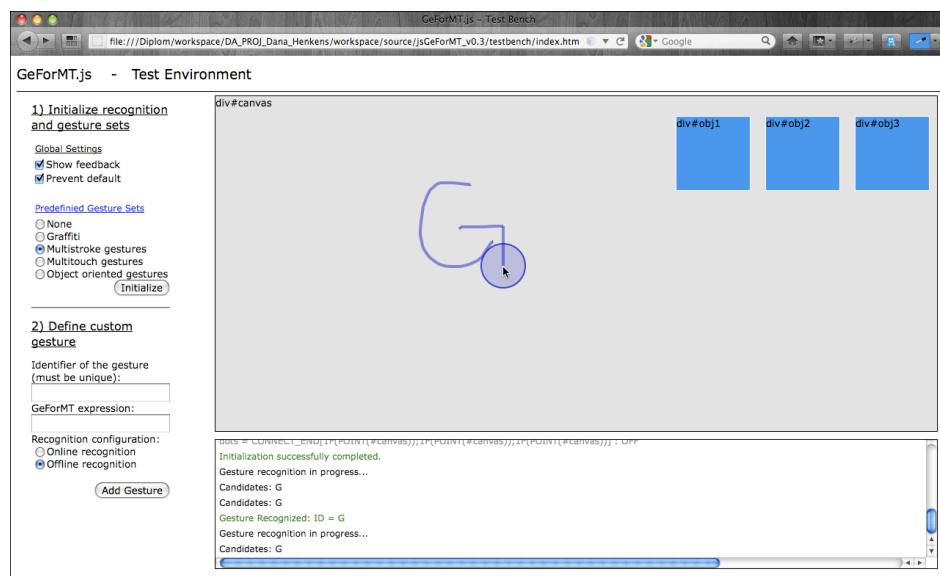


Abbildung 5.10: Web-Interface der Testumgebung

Vordefinierte Gesten zum Testen, Set 1: Einzelpfad-Gesten

Die vordefinierten Gesten verdeutlichen die vielfältigen Anwendungsfälle, die in der Implementierung des Frameworks berücksichtigt wurden. Aus diesem Grund werden die Gesten-Sets im Folgenden vorgestellt. Exemplarisch für Einzelpfad-Gesten sind in der ersten Auswahl Gesten aus dem *Graffiti*-Symbolalphabet (A-Z) definiert. Sie wurden bereits in Kapitel 3.4.4 vorgestellt. (vgl. Abb. 3.26)

Set 2: Zusammengesetzte Gesten

Es folgt eine Auswahl zusammengesetzter Gesten (Option *Multistroke gestures*) wie sie in Abbildung 5.11 skizziert sind. Ihre Erkennung basiert hauptsächlich auf der Differenzierung von sequentiellen Kompositionsoperatoren in Kapitel 4.2.4, sowie der Implementierung der horizontalen und vertikalen Nachbarschaftsbeziehung.

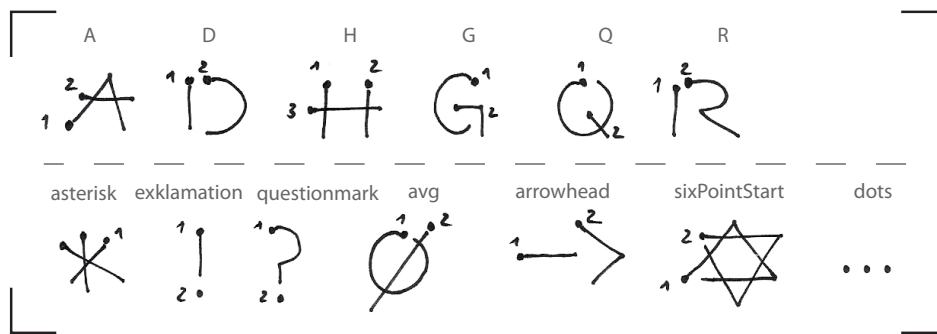


Abbildung 5.11: Skizzierung der vordefinierten, sequentiell komponierten Gesten

Abbildung 5.12 skizziert die repräsentativen Multitouch-Gesten (Option *Multitouch gestures*, deren Erkennung sich vorallem auf einer Unterscheidung der vorgeschriebenen Relationen SYNC, SPLIT und JOIN stützt.

Set 3: *Multitouch-Gesten*

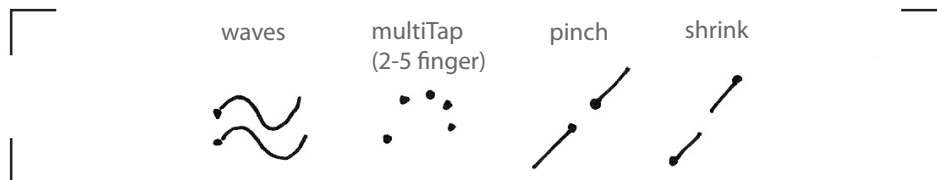


Abbildung 5.12: Skizzierung der definierten Menge synchron ausgeführter Gesten

Das letzte Set definiert objektorientierte Gesten (Option *Object oriented gestures*). Da sie zum Großteil zur Eingabezeit erkannt werden, wird an dieser Stelle ein spezifisches Feedbackverhalten implementiert, um die Erkennung dieser Gesten besser testen zu können. Vier objektorientierte Gesten sind in diesem Set definiert: Das Antippen (*tapObject*), Verschieben (*moveObject*), Skalieren (*moveObject*) und die Auswahl von einem oder mehreren Objekten mit einer LASSO-Geste. Abbildung 5.13 veranschaulicht das Verhalten der Objekte bei Erkennung der letzten drei Gesten.

Set 4: *Objektorientierte Gesten*

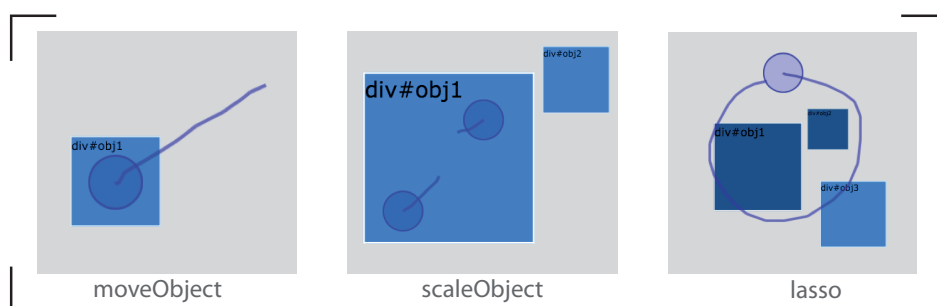
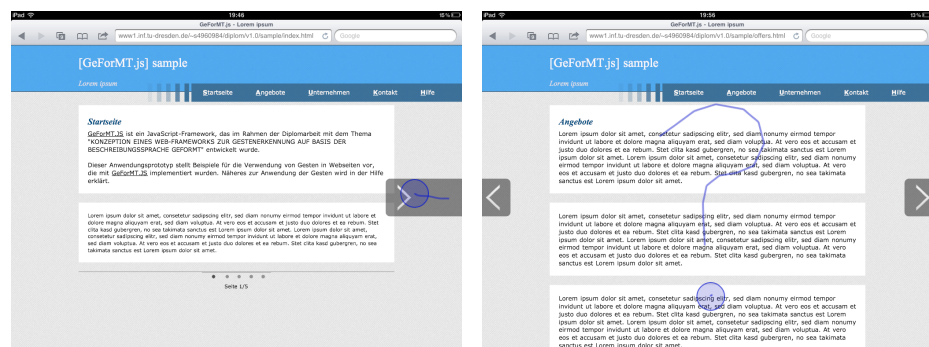


Abbildung 5.13: Skizzierung der objektorientierter Gesten zum Verschieben, Skalieren und Selektieren

Details zum Bestand der Gesten und deren Ausführungseigenschaften können der jeweiligen formalen Definition entnommen werden. Diese werden in der Testumgebung bei der Initialisierung in der Konsole angezeigt.

5.3.4 Anwendungsprototyp

Für die Implementierung eines Anwendungsprototypen wird das in Kapitel 4.1 beschriebene Anwendungsszenario aufgegriffen. Die darin vorgestellten Konzepte der gestenbasierten Navigation werden exemplarisch in einer einfach gehaltenen, informativen Web-Seite umgesetzt. Optimiert ist die Implementierung für die Bedienung auf einem iPad (Version 4.3.3). Das geräteübergreifende Framework-Konzept erlaubt die Anwendung der umgesetzten Gesten jedoch auch für mausbasierte Eingabesysteme. In den Abbildungen 5.14a und 5.14b wird der Aufbau des Web-Anwendung deutlich. Angelehnt an eine simple Unternehmenspräsentation, wurden insgesamt fünf Seiten umgesetzt, die wie gewohnt über das Hauptmenü in der oberen Leiste aufgerufen werden können.



(a) Blättern-Funktion im Anwendungsprototypen
(GeForMT-Definition:
 $1F(LINE\_W(\#nextPageLabel)):0N$)

(b) Anwendung symbolischer Gesten
(GeForMT-Definition des Fragezeichen-Symbols:
 $1F(SEMICIRCLE\_SE, LINE\_S); 1F(POINT):OFF$)

Abbildung 5.14: Interface des Anwendungsprototypen

*Führung durch die
Web-Anwendung
durch Blättern*

Alternativ können diese Seiten über die seitlichen Pfeilmarkierungen durchgeblättert werden. Die darauf auszuführende Geste ist der Bedienung eines Lesezeichens in einem Buch nachempfunden. Entsprechend müssen die Marken ausreichend weit vom Seitenrand weggezogen werden, um auf die verlinkte Seite zu gelangen (vgl. Abbildung 5.14a). Die Blättern-Funktion bezieht sich in diesem Fall auf die Hauptseiten. Die Reihenfolge orientiert sich an der Auflistung der Menüpunkte. Denkbar wäre diese Art der Nutzerführung vorallem auf möglichen Unterseiten in Anlehnung an eine Strukturierung der Informationen in Kapiteln.

*Symbolische Gesten
und Navigation durch
Browser-History*

Aufbauend auf dem Konzept von Access Keys (vgl. Kapitel 4.1) können die Seiten alternativ über assoziierte Symbole angesteuert werden. Hierzu wurden die Anfangsbuchstaben der Menüpunkte als Gesten implementiert. Weitere Gesten wie beispielsweise ein Fragezeichen für den Hilfeaufruf wurden ebenso mit aufgenommen. (vgl. Abbildung 5.14b) Darüberhinaus wird das Blättern in der Browser-History mit einer schleifenartigen Bewegung ermöglicht, wie es als Adaption von Browser-Funktionen in Kapitel 4.1 angedacht wurde. Näheres zu der Bewegungsausführung, sowie Formalisierung der Gesten mit GeForMT ist auf der Hilfeseite der Anwendung beschrieben. Bei Eingaben, die vom Framework nicht als Geste erkannt werden konnten, wird eine Fehlermeldung mit Verweis auf die Hilfe ausgegeben.

*Herausforderungen für
Web-Entwickler*

In der Implementierung des Anwendungsprototypen werden generelle Probleme bei der Umsetzung von Gesten in Web-Anwendungen deutlich. Die Herausforderung besteht darin, dass Web-Browser standardmäßig Gesten implementieren, die mit anwendungsspezifischen Gesten in Konflikt treten. Das standardmäßige Verschieben und Skalieren einer Seite zur Eingabezeit steht in Konflikt mit den symbolischen

Gesten dieses Anwendungsprototypen. Verallgemeinernd besteht dieser Konflikt zwischen Online- und Offline-Gesten. Online-Gesten werden aufgrund des früheren Interpretationszeitpunkt priorisiert. Das Gestenerkennungs-Framework kann so konfiguriert werden, dass das Standverhalten des Browsers unterdrückt wird (vgl. Kapitel 5.3.2). Daraus folgt jedoch auch, dass das berührungsbasierte Scrollen, wie auch Skalieren der Web-Dokumente nicht mehr möglich ist. Der Anwendungsprototyp implementiert daher eine adaptierte Variante einer DRAG-Geste basierend auf GeForMT. Das Scrollen der Seite ist damit an den seitlichen Rändern des Dokumentes möglich. Gemäß einer Größenanpassung von Inhalten kann die Schriftgröße einzelner Textpassagen mit einer PINCH- bzw. SHRINK-Geste (vgl. Kapitel 3.4.4) optimiert werden.

Mögliche Erweiterungen des Anwendungsbeispiels sind gestenbasierte Kontextmenüs. Zum Beispiel könnten Ringmenüs, die durch das Halten mehrerer Finger auf dem Touch-Oberfläche oder auch durch eine kreisförmige Gesten aufgerufen werden die statische Menüleiste ersetzen und so mehr Platz für Inhalte geben. Erweiternd könnten Gesten wie das Abhaken, Ankreuzen, Unterstreichen oder Umkreisen für die Selektion verbreiteter interaktiver Elemente wie Links oder Auswahloptionen in Formularen und Dialogen umgesetzt werden. Auch das Scrollen und Blättern auf elementaren Komponenten des Web-Interface könnte mehr fokussiert werden, um beispielsweise Frames in touchbasierten Web-Anwendungen zu adaptiert. Das GeForMT-Framework bietet ausreichend Potenzial um diese Konzepte umzusetzen.

Erweiterungspotenzial

6 ZUSAMMENFASSUNG UND FAZIT

Ziel der Diplomarbeit war es, ein Framework auf Basis der Gestenbeschreibungssprache GeForMT zu entwerfen und im Anwendungsbereich von webbasierten Umgebungen prototypisch zu implementieren. Hierfür wurden zunächst gegenwärtige softwaretechnische Grundlagen der Web-Entwicklung erarbeitet, sowie Eigenschaften zweidimensionaler Gesten behandelt. Unter dem Gesichtspunkt einer geräteübergreifenden Anwendung des Frameworks lag der Schwerpunkt der technischen Grundlagenbehandlung vor allem auf der Eingabedatenverarbeitung und möglichen Ansätzen der Generalisierung von Maus-, Touch- und Stifteingaben.

Diesbezüglich wurden die Interaktionstechniken für eine webkonforme Gestenformalisierung gegenübergestellt und deren Abbildbarkeit analysiert. Durch die Berücksichtigung dieser Aspekte in der Framework-Implementierung werden sowohl klassische mausbasierte Desktop-Systeme, als auch berührungsbasierte Eingabesysteme unterstützt. Gemäß einer breiten und vielfältigen Unterstützung unabhängig von der eingesetzten Hardware wurde ergänzend die Verarbeitung des TUIO-Protokolls in Betracht gezogen. Des Weiteren wurden im Kapitel der verwandten Arbeiten bestehende Web-Frameworks untersucht und festgestellt, dass sich die meisten Lösungen auf eine Menge verbreiteter Gesten festlegt und häufig auf eine Emulation plattformspezifischer mobiler User-Interfaces abzielen.

Das umgesetzte GeForMT-Framework konzentriert sich hingegen auf eine Erkennungsmethodik unspezifizierter Gesten. Dem Web-Entwickler wird damit ein Instrument angeboten, das ihm größtmögliche gestalterische Freiheiten gewährt und den Implementierungsaufwand von Gesten auf ein Minimum reduziert. Dies wird nicht zuletzt durch die verständliche und leicht zu erlernende Beschreibungssprache GeForMT unterstützt, die den Kern der Framework-API bildet.

Hinsichtlich einer nahtlosen Integration formaler Gestendefinitionen in Web-Dokumenten wurden in der Synthese dieser Arbeit Möglichkeiten der Schnittstellenimplementierung diskutiert, die die Interoperabilität zwischen grafischen Interface-Komponenten und Aktionsbeschreibung herstellen. Hierbei wurde auf bewährte Softwaretechniken, wie beispielsweise die Verwendung von Selektoren, zurückgegriffen. Darüber hinaus wurde sich auch mit generellen Ansätzen der Formalisierung auseinandergesetzt und daraus resultierend Erweiterungsmöglichkeiten vorgestellt, die in erster Linie Uneindeutigkeiten auflösen sollen und darüber hinaus die Beschreibungsfähigkeit der Sprache erhöhen.

Das Endprodukt dieser Arbeit ist ein modulares JavaScript-Framework, das sich durch seine Erweiterbarkeit und seinen plattformübergreifenden, flexiblen Einsatz auszeichnet. Aufbauend auf der Bewertung bestehender Methoden der Gestenerkennung hinsichtlich ihrer Realisierbarkeit in einer webbasierten Software und Anwendbarkeit auf formale Gestendefinitionen wurde sich für die Erkennungsmethodik des

\$1-Recognizer entschieden. Bewährt hat sich dieser Algorithmus insbesondere in Bezug auf seine Erweiterbarkeit. Es konnte gezeigt werden, dass über die Standard-Implementierung hinaus auch zusammengesetzte Gesten, Multitouch-Gesten und sogar Online-Gesten erkannt werden können. Details zum Entwurf der Framework-Komponenten, sowie die allgemeine Anwendung des API wurden im Kapitel der Konzeption und Implementierung beschrieben.

Zu Testzwecken wurde eine Web-Anwendung entwickelt, in der die Funktionsweise des Frameworks nachvollzogen werden kann. Die Testumgebung beinhaltet sowohl beispielhafte, vordefinierte Gesten-Sets, als auch Funktionen, mit denen sich weitere Gesten definieren lassen. Entwickeln wird damit auch ein Werkzeug angeboten, mit dem Gesten dahingehend überprüft werden können, ob sie harmonisieren oder miteinander in Konflikt treten. Die Arbeit schließt mit einem Anwendungsprototypen auf einem Multitouch-Tablet ab, der Perspektiven für den Einsatz des Gestenerkennungsframeworks in einfachen informativen Web-Seiten aufzeigt.

6.1 AUSBLICK

Mögliche Erweiterungen der Gestenformalisierung speziell für Web-Anwendungen wurden in Kapitel 4.2.10 zusammengefasst und sind auf hinreichende Bedeutung für die Weiterentwicklung der Beschreibungssprache zu prüfen. In diesem Kapitel werden Perspektiven für die Weiterführung des Themas gegeben. Im Sinne einer kritischen Auseinandersetzung mit dem Framework-Konzept werden an dieser Stelle auch optimierbare Aspekte herausgestellt.

Weitere Systemtests vornehmen

Ein erster Diskussionspunkt ist die Verallgemeinerung, dass das Framework aufbauend auf den Standards des W3C als plattformunabhängig gilt. Letztlich ist die Funktion des Frameworks nur dann gewährleistet, wenn dessen Ausführungsumgebung, sprich der Web-Browser, die Empfehlungen des W3C korrekt umsetzt. Trotz weitreichender Verbreitung der Standards sind browserspezifische Differenzen und Eigenheiten nicht auszuschließen. Insbesondere von der Umsetzung von Standards, die sich noch in ihrer Entwurfsphase befinden ist nicht unmittelbar auszugehen. Entwickelt und getestet wurde das Framework im Mozilla Firefox, der aktuell in der Version 10.0 vorliegt und im integrierten Safari-Browser des iPad 4.3 von Apple. Im nächsten Schritt sind weitere mobile, als auch desktopbasierte Web-Browser zu testen, um Aussagen zur Unterstützung des Frameworks zu konkretisieren.

Performanzoptimierung

In Anbetracht der umfangreichen Optionen, die aus dem Formalisierungsansatz hervorgehen und interpretiert werden müssen, stellt die Anforderung an eine performante Verarbeitung eine besondere Herausforderung dar. Auf den getesteten Systemen wird die Verarbeitungsgeschwindigkeit als akzeptabel erachtet, dennoch besteht Optimierungsbedarf insbesondere für leistungsschwächere mobile Endgeräte. Eine Möglichkeit, um die Performanz zu erhöhen, wurde in den Grundlagen dieser Arbeit aufgezeigt. Dabei handelt es sich um den Einsatz von *Web Worker* (siehe Kapitel 2.3.5), mit denen Verarbeitungsprozesse parallelisiert werden können. Aufgrund der geringen Unterstützung dieser Methode auf den eingesetzten Testsystemen wurde sie in der Implementierung nicht weiter berücksichtigt. Weiterhin wurde in Kapitel 3.3.3 eine Optimierung des \$1-Berechnungsverfahren auf Basis von Vektoren nach [Li, 2010b] vorgestellt. Der Ansatz wurde als Option berücksichtigt und implementiert, zeigte aber Schwächen bei der Erkennung eindimensionaler Gesten auf und konnte daher nicht angewendet werden. In der Weiterentwicklung des Frameworks sollten diese Ansätze vorallem hinsichtlich des erhöhten Rechenaufwands bei

der Erkennung von Multitouch-Gesten und der Visualisierung von Gestenpfaden in Betracht gezogen werden.

Hinsichtlich der Optimierung ist an dieser Stelle auf eine Schwäche des \$1-Erkennungsverfahrens hinzuweisen. Es wurde erklärt, dass die Berechnungsmethode eine Normierung der Punktkoordinaten von Testmuster und Referenzmuster voraussetzt. Dies führt dazu, dass die räumlichen Dimensionen nicht mehr berücksichtigt werden können. Bei der Definition von Einzelpfad-Gesten kann diese Schwäche durch Wiederholungen von Gestenatomen ausgeglichen werden. Ein Beispiel hierfür ist die Gestendefinition `LINE_NE, LINE_S, LINE_S` für die Beschreibung der Ziffer 1. Dies ist jedoch nicht auf die Komposition von kreisförmigen Gesten anwendbar, wie zum Beispiel eine spiralförmige Bewegungsform, die in Kapitel 4.2.4 in Betracht gezogen wurde. Deutlich wird das Problem vor allem in zusammengesetzten Gesten, deren Pfade sukzessiv verarbeitet werden. Symbolhafte Gesten wie beispielsweise D (beschrieben durch `LINE_S; SEMICIRCLE_S`) und P (beschrieben durch `LINE_S, LINE_S; SEMICIRCLE_S`) können nicht unterschieden werden. Eine Auflösung dieser Problematik erfordert Erweiterungen, die die Länge und den Umfang von Pfaden in die Erkennung einbeziehen.

*Grenzen des
Erkennungsverfahrens*

In Hinblick auf eine vollständige Implementierung der in GeForMT definierten Sprachoptionen besteht ebenso noch Erweiterungsbedarf. Eine Unterscheidung zwischen asynchroner und synchroner Komposition von Gesten wird beispielsweise nicht umgesetzt und erfordert Erweiterungen, die zeitliche Dimensionen berücksichtigen. In Ermangelung einer Zeiterfassung wird in dieser Version auch das längere Halten eines Kontaktes (`HOLD`) nicht erkannt.

*Erweiterung um eine
Zeiterfassung*

Die Visualisierung von Berührungen und Bewegungspfaden wird im Framework weitestgehend unabhängig vom Ausführungskontext definierter Gesten implementiert. In weiterführenden Arbeiten könnten Feedback und Geste stärker in Beziehung gesetzt werden, um in der Art der Visualisierung zu differenzieren. Ein Ansatz in diese Richtung zeigt das Verfahren bei zusammengesetzten Gesten, in dem die Erwartung weiterer Eingaben in die Visualisierungslogik einfließen. Eine Differenzierung bezüglich der Eingabetechnik ist ebenfalls abzuwägen. Darüber hinaus fehlt es noch an Einstellungsmöglichkeiten, um auf die Gestaltung des primitiven Feedbacks Einfluss zu nehmen. Um den Nutzer in der Bedienung gestenbasierter Anwendungen zu unterstützen sind des Weiteren gestalterische Aspekte von Web-Bedienelementen zu fokussieren. Dies beinhaltet insbesondere Feed-Forward-Mechanismen, die dem Nutzer Hilfestellung zu anwendbaren Gesten anbieten.

*Feedback und
Feed-Forward*

Motiviert wurde die Entwicklung des Gestenerkennungs-Frameworks durch die Idee der Unterstützung von Web-Programmierern bei der Gestaltung neuer Bedienkonzepte. Auf Basis dieser Grundlagenschaffung kann der Fokus auf das Design von gestenbasierten Web-Anwendungen gelegt werden. Hinsichtlich einer Weiterführung des Themas sind konkrete Anwendungen zu entwickeln. Kleinere Anwendungsbeispiele bezüglich der Adaption einer Web-Seite auf einem Multitouch-Tablet wurden in dieser Arbeit vorgestellt und könnten in einem komplexeren Anwendungsszenario aufgegriffen werden. Interessant wäre vor allem die Kombination des GeForMT-Frameworks mit grafischen UI-Bibliotheken, welche im Folgenden noch zu testen ist. Als Anregung für einen konkreten Anwendungsfall könnten beispielsweise Webseiten für öffentliche Verkehrsmittel entwickelt werden. Insbesondere eine Routenplanung auf Basis einer Karte wie beispielsweise *Google Maps* oder einem Verkehrsplan bietet ausreichend Potenzial für ein gestenbasiertes Interaktionsdesign. Doch auch im Bereich der prototypischen Software-Entwicklung kann sich das Framework als nützliches Werkzeug erweisen. Im Sinne einer kundenorientierten Entwicklung proprietärer Software, wie beispielsweise einer App auf einem iPad, wäre eine rudimentäre Implementierung der Benutzerschnittstelle (sogenannte Mock-ups) auf

*Entwicklung konkreter
Anwendungen*

Basis von Web-Technologien, ergänzt um die Gestenformalisierung mit weniger Implementierungsaufwand und Programmierwissen verbunden.

Dreidimensionale Gestenerkennung

Basierend auf weiteren in Arbeit befindlichen W3C-Standards wie beispielsweise der *DeviceOrientation*-Spezifikation [W3C, 2011b] und kommenden Schnittstellen für die Kameraauswertung [W3C, 2011d] ist eine Weiterentwicklung von GeForMT hinsichtlich der dreidimensionalen Gestenerkennung denkbar. Für die Formalisierung räumlicher Gesten könnten Ansätze aus [Henkens, 2011] aufgegriffen werden. Die Möglichkeiten für die Weiterführung der Arbeit sind damit sehr vielfältig.

LITERATURVERZEICHNIS

- [Anthony und Wobbrock 2010] Anthony, Lisa; Wobbrock, Jacob O.: A lightweight multistroke recognizer for user interface prototypes. In: *Proceedings of Graphics Interface 2010*. Toronto, Ont., Canada, Canada : Canadian Information Processing Society, 2010 (GI '10), S. 245–252. – URL: <http://portal.acm.org/citation.cfm?id=1839214.1839258>. – ISBN 978-1-56881-712-5
- [Apers 2011] Apers, Chris: *WebApp.NET - A web application micro-framework*. 2011. – URL: <http://webapp-net.com/>. (Zugriffsdatum: 10.02.2012)
- [Apple 2009a] Apple: *iPhone User Guide (Online-Benutzerhandbuch für das iPhone 3.0)*. 2009. – URL: <http://help.apple.com/iphone/3/mobile/>. (Zugriffsdatum: 10.02.2012)
- [Apple 2009b] Apple: *Magic Mouse*. Okt. 2009. – URL: <http://www.apple.com/de/magicmouse/>. (Zugriffsdatum: 14.10.2011)
- [Apple 2010] Apple: *Magic Trackpad*. Juli 2010. – URL: <http://www.apple.com/magictrackpad/>. (Zugriffsdatum: 14.10.2011)
- [Apple 2011a] Apple: *Safari Web Content Guide*. Dez. 2011. – URL: <http://developer.apple.com/library/safari/#documentation/AppleApplications/Reference/SafariWebContent>. (Zugriffsdatum: 28.09.2011)
- [Apple 2011b] Apple: *Webapps: What are WebApps?* 2011. – URL: <http://www.apple.com/de/webapps/whatarewebapps.html>. (Zugriffsdatum: 04.11.2011)
- [Apple 2012] Apple: *Apple iOS*. 2012. – URL: <http://www.apple.com/de/ios/>. (Zugriffsdatum: 11.02.2012)
- [Bieh 2008] Bieh, Manuel; Press, Galileo (Hrsg.): *Mobiles Webdesign - Konzeption, Gestaltung, Entwicklung*. 2008. – ISBN 978-3-8362-1153-6
- [Bimber et al. 2000] Bimber, Oliver; Encarnação, L. M.; Stork, André: A multi-layered architecture for sketch-based interaction within virtual environments. In: *Computers Graphics* (2000), S. 851–867
- [Blas 2011] Blas, Kin: *The Current State of (Touch) Events*. März 2011. – URL: <http://blogs.adobe.com/adobeandjquery/2011/03/07/the-current-state-of-touch-events/>
- [Blickenstorfer 1995] Blickenstorfer, C. H.: Graffiti: Wow! Pen Computing Magazine. In: *Pen Computing Magazine* (1995), Jan., S. 30–31
- [Cailliau et al. 2009] Cailliau, Pieter; Demeulemeester, Aljosha; Deprez, Jean-Luc; Schepper, Kristof D.; Frederix, Jan; Mees, Pieter; Seuryneck, Jense; Bulcke, Dieter V. den; Vittorelli, Michael: *Pooky: PookyTouch and jPooky*. 2009. – URL: <http://pooky.sourceforge.net/>. (Zugriffsdatum: 11.02.2012)

- [Calhoun 2009] Calhoun, David: *PastryKit: digging into an Apple Pie*. Dez. 2009. – URL: <http://davidbcalhoun.com/2009/pastrykit-digging-into-an-apple-pie>. (Zugriffsdatum: 15.11.2011)
- [Carstensen 2011] Carstensen, Brian: *Phantom Limb*. Jan. 2011. – URL: <http://www.vodori.com/blog/phantom-limb.html>. (Zugriffsdatum: 30.09.2011)
- [Charland und LeRoux 2011] Charland, Andre; LeRoux, Brian: Mobile Application Development: Web vs. Native. In: *Queue* 9 (2011), April, S. 20:20–20:28. – URL: <http://doi.acm.org/10.1145/1966989.1968203>. – ISSN 1542-7730
- [Cheng et al. 2009] Cheng, Kelvin; Itzstein, Benjamin; Sztajer, Paul; Rittenbruch, Markus: *A unified multi-touch multi-pointer software architecture for supporting collocated work on the desktop*. 2009
- [Cooper et al. 2007] Cooper, Alan; Reimann, Robert; Cronin, David: *About Face 3: The Essentials of Interaction Design*. 3rd. Wiley, 5 2007. – ISBN 9780470084113
- [Damaraju und Kerne 2008] Damaraju, Sashikanth; Kerne, Andruid: *Multitouch Gesture Learning and Recognition System*. 2008. – URL: <http://www.sriramsubramanian.com/tabletop2008/images/paper244.pdf>. (Zugriffsdatum: 11.02.2012)
- [Debreuil 2001] Debreuil, Robin: *Just What is Object-Oriented Programming? - The Prototype*. 2001. – URL: http://www.debreuil.com/docs/ch01_Prototype.htm. (Zugriffsdatum: 20.10.2011)
- [Dell 2011] Dell: *DELL Monitors: Dell SX2210T 55cm (21.5")W Multi-Touch Monitor with Webcam*. 2011. – URL: <http://www.dell.com/ae/business/p/dell-sx2210t/pd>. (Zugriffsdatum: 14.10.2011)
- [Deveria 2011a] Deveria, Alexis: *When can I use pointer-events? - Compatibility table for support of pointer-events in desktop and mobile browsers*. Oct. 2011. – URL: <http://caniuse.com/pointer-events>. (Zugriffsdatum: 10.02.2012)
- [Deveria 2011b] Deveria, Alexis: *When can I use Web Workers? - Compatibility table for support of Web Workers in desktop and mobile browsers*. Sept. 2011. – URL: <http://caniuse.com/webworkers>. (Zugriffsdatum: 24.10.2011)
- [Dojo 2011] Dojo: *Dojo mobile*. Juli 2011. – URL: <http://dojotoolkit.org/features/mobile>. (Zugriffsdatum: 30.09.2011)
- [Dorau 2011] Dorau, Rainer: *Emotionales Interaktionsdesign: Gesten und Mimik ineraktiver Systeme (X.Media.Press)*. 1., st Edition. Springer, Berlin, 5 2011. – ISBN 9783642031007
- [Dumke et al. 2003] Dumke, Reiner; Lothar, Mathias; Wille, Cornelius; Zbrog, Fritz: *Web Engineering*. 1., Aufl. Pearson Studium, 9 2003. – ISBN 9783827370808
- [Echtler und Klinker 2008] Echtler, Florian; Klinker, Gudrun: A multitouch software architecture. In: *Proceedings of the 5th Nordic conference on Human-computer interaction: building bridges*. New York, NY, USA : ACM, 2008 (NordiCHI '08), S. 463–466. – URL: <http://doi.acm.org/10.1145/1463160.1463220>. – ISBN 978-1-59593-704-9
- [Echtler et al. 2010] Echtler, Florian; Klinker, Gudrun; Butz, Andreas: Towards a unified gesture description language. In: *Proceedings of the 13th International Conference on Humans and Computers*. Fukushima-ken, Japan, Japan : University of Aizu Press, 2010 (HC '10), S. 177–182. – URL: <http://portal.acm.org/citation.cfm?id=1994486.1994528>

- [ELO TouchSystems 2011] ELO TouchSystems: *ELO Touch Systems: Touchscreen-Technologie – Möglichkeiten*. 2011. – URL: <http://www.elotouch.de/Produkte/detech2.asp>. (Zugriffsdatum: 17.10.2011)
- [Forlines et al. 2007] Forlines, Clifton; Wigdor, Daniel; Shen, Chia; Balakrishnan, Ravin: Direct-touch vs. mouse input for tabletop displays. In: *Proceedings of the SIGCHI conference on Human factors in computing systems*. New York, NY, USA : ACM, 2007 (CHI '07), S. 647–656. – URL: <http://doi.acm.org/10.1145/1240624.1240726>. – ISBN 978-1-59593-593-9
- [Geißler 2001] Geißler, Jörg: *Design und Implementierung einer stiftzentrierten Benutzungsoberfläche*. Darmstadt, Technische Universität Darmstadt, Dissertation, April 2001
- [Gibson 2011] Gibson, Alex: *WKTouch*. Sept. 2011. – URL: <https://github.com/alexgibson/WKTouch>. (Zugriffsdatum: 30.09.2011)
- [Goldberg und Richardson 1993] Goldberg, David; Richardson, Cate: Touch-typing with a stylus. In: *Proceedings of the INTERACT '93 and CHI '93 conference on Human factors in computing systems*. New York, NY, USA : ACM, 1993 (CHI '93), S. 80–87. – URL: <http://doi.acm.org/10.1145/169059.169093>. – ISBN 0-89791-575-5
- [Google 2011] Google: *Offizieller Webaufttritt der Android-Entwickler*. 2011. – URL: <http://www.android.com/>. (Zugriffsdatum: 22.01.2011)
- [Görg 2009] Görg, Martin: *A Framework for Abstract Representation and Recognition of Gestures in Multi-Touch Applications*, Technische Universität Berlin, Diplomarbeit, Okt. 2009
- [Hallman 2011] Hallman, Daniel: `<script src="????.js">` - Vergleich von JavaScript Frameworks. Publikation unter <http://isuu.com>. Feb. 2011. – URL: http://issuu.com/visualcosmos/docs/artikel_-_js_frameworks. (Zugriffsdatum: 04.11.2011)
- [Hayes 2010] Hayes, Paul: *CSS 3D cube with touch gestures, click and drag*. Sept. 2010. – URL: <http://www.paulrhayes.com/2010-09/3d-css-cube-ii-touch-gestures-click-and-drag/>. (Zugriffsdatum: 04.10.2011)
- [Henkens 2011] Henkens, Dana: *Kombination gestenbasierter Interaktion in multimodalen Anwendungen*, Technische Universität Dresden, Bakkalaureat, 2011
- [Henzen 2010] Henzen, Christin: *Prototypische Konzeption eines Frameworks basierend auf einer formalen Beschreibungssprache basierend auf einer formalen Beschreibungssprache*, Technische Universität Dresden, Diplom, 2010
- [Hewitt 2011] Hewitt, Joe: *iUI - web framework for smartphones high-end devices*. 2011. – URL: www.iui-js.org/. (Zugriffsdatum: 30.09.2011)
- [Hinrichs und Carpendale 2011] Hinrichs, Uta; Carpendale, Sheelagh: Gestures in the wild: studying multi-touch gesture sequences on interactive tabletop exhibits. In: *Proceedings of the 2011 annual conference on Human factors in computing systems*. New York, NY, USA : ACM, 2011 (CHI '11), S. 3023–3032. – URL: <http://doi.acm.org/10.1145/1978942.1979391>. – ISBN 978-1-4503-0228-9
- [Holz und Baudisch 2011] Holz, Christian; Baudisch, Patrick: Understanding touch. In: *Proceedings of the 2011 annual conference on Human factors in computing systems*. New York, NY, USA : ACM, 2011 (CHI '11), S. 2501–2510. – URL: <http://doi.acm.org/10.1145/1978942.1979308>. – ISBN 978-1-4503-0228-9
- [HP 2011] HP: *HP - Produkte und Devices: TouchPad*. 2011. – URL: <http://h41112.www4.hp.com/promo/webos/de/de/tablet/touchpad.html>. (Zugriffsdatum: 25.01.2012)

- [Hunter et al. 2010] Hunter, Jason; Driscoll, Jim; Murray, Greg: *The HTML Pad*. 2010. – URL: <http://htmlpad.wordpress.com>. (Zugriffsdatum: 04.10.2011)
- [jQuery 2011] jQuery: *jQuery Mobile: Touch-Optimized Web Framework for Smartphones Tablets*. Okt. 2011. – URL: <http://jquerymobile.com/>. (Zugriffsdatum: 14.11.2011)
- [Kaltenbrunner 2011] Kaltenbrunner, Martin: *TUIO*. 2011. – URL: <http://www.tuio.org>. (Zugriffsdatum: 20.01.2012)
- [Kaltenbrunner und Bencina 2009] Kaltenbrunner, Martin; Bencina, Ross: *reactIVision 1.4 - a toolkit for tangible multi-touch surfaces*. Mai 2009. – URL: <http://reactivision.sourceforge.net/>. (Zugriffsdatum: 07.11.2011)
- [Kammer et al. 2011] Kammer, Dietrich; Henzen, Christin; Keck, Mandy; Taranko, Severin: GeForMT: Gestenformalisierung für Multitouch. In: Groh, Rainer (Hrsg.); Zavesky, Martin (Hrsg.): *Wieder mehr Sehen! - Aktuelle Einblicke in die Technische Visualistik*, TUDpress, 2011
- [Kammer et al. 2010] Kammer, Dietrich; Wojdziak, Jan; Keck, Mandy; Groh, Rainer; Taranko, Severin: Towards a formalization of multi-touch gestures. In: *ACM International Conference on Interactive Tabletops and Surfaces*. New York, NY, USA : ACM, 2010 (ITS '10), S. 49–58. – URL: <http://doi.acm.org/10.1145/1936652.1936662>. – ISBN 978-1-4503-0399-6
- [Kaneda und Stark] Kaneda, David; Stark, Jonathan: *jQTouch - A jQuery plugin for mobile web development on the iPhone, Android, iPod Touch, and other forward-thinking devices*. – URL: <http://jqtouch.com/>. (Zugriffsdatum: 14.11.2011)
- [Khandkar und Maurer 2010a] Khandkar, Shahedul H.; Maurer, Frank: *A Domain Specific Language to Define Gestures for Multi-Touch Applications*. Oct. 2010. – URL: http://ase.cpsc.ucalgary.ca/uploads/Publications/KhandkarDSM2010_v2.pdf. (Zugriffsdatum: 11.02.2012)
- [Khandkar und Maurer 2010b] Khandkar, Shahedul H.; Maurer, Frank: *GestureToolkit - A toolkit to simplify the multi-touch application development and testing complexities*. Okt. 2010. – URL: <http://gesturetoolkit.codeplex.com/>. (Zugriffsdatum: 16.11.2011)
- [Koch 2010] Koch, Peter-Paul: *Touch table*. 07. 2010. – URL: <http://www.quirksmode.org/mobile/tableTouch.html>. (Zugriffsdatum: 28.09.2011)
- [Lessiak 2008] Lessiak, Marius: *Interaktionsdesign in Webapplikationen: Gestalten, Bewerten, Umsetzen*. Vdm Verlag Dr. Müller, 3 2008. – ISBN 9783836464505
- [Li 2010a] Li, Yang: Gesture search: a tool for fast mobile data access. In: *Proceedings of the 23rd annual ACM symposium on User interface software and technology*. New York, NY, USA : ACM, 2010 (UIST '10), S. 87–96. – URL: <http://doi.acm.org/10.1145/1866029.1866044>. – ISBN 978-1-4503-0271-5
- [Li 2010b] Li, Yang: Protractor: a fast and accurate gesture recognizer. In: *Proceedings of the 28th international conference on Human factors in computing systems*. New York, NY, USA : ACM, 2010 (CHI '10), S. 2169–2172. – URL: <http://doi.acm.org/10.1145/1753326.1753654>. – ISBN 978-1-60558-929-9
- [Majda 2010] Majda, David: *PEG.js - Parser Generator for JavaScript*. 2010. – URL: <http://pegjs.majda.cz/>. (Zugriffsdatum: 06.11.2011)
- [Moyle und Cockburn 2003] Moyle, Michael; Cockburn, Andy: The design and evaluation of a flick gesture for 'back' and 'forward' in web browsers. In: *Proceedings of the Fourth Australasian user interface conference on User interfaces 2003 - Volume 18*. Darlinghurst, Australia, Australia : Australian

- Computer Society, Inc., 2003 (AUIC '03), S. 39–46. – URL: <http://portal.acm.org/citation.cfm?id=820086.820097>. – ISBN 0-909925-96-8
- [Mozilla 2011] Mozilla: *Mozilla Developer Network*. 2011. – URL: <https://developer.mozilla.org>. (Zugriffsdatum: 28.09.2011)
- [Mozilla 2012] Mozilla: *Mozilla Firefox Browser*. 2012. – URL: <http://www.mozilla.org/de/firefox>. (Zugriffsdatum: 10.02.2012)
- [Murray 2010] Murray, Greg: *iPad Web Development*. 2010. – URL: <http://gregmurray.org/ipad/>. (Zugriffsdatum: 04.10.2011)
- [Nygård 2010] Nygård, Espen S.: *Multi-touch Interaction with Gesture Recognition*, Norwegian University of Science and Technology, Department of Computer and Information Science, Diplomarbeit, June 2010. – URL: <http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=0FA32BF38832DC34020370E01BCBC47A?doi=10.1.1.175.422&rep=rep1&type=pdf>. (Zugriffsdatum: 11.02.2012)
- [Poirier 2005] Poirier, Franck: *Glyph: A New Stroke Alphabet for Stylus-Based or Key-Based Text Entry*. 2005. – URL: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.89.6001&rep=rep1&type=pdf>. (Zugriffsdatum: 11.02.2012)
- [Pojer 2011] Pojer, Christoph: *MooTools Mobile*. 2011. – URL: <http://mootools.net/forged/p/mobile>. (Zugriffsdatum: 15.11.2011)
- [Qooxdoo 2011] Qooxdoo: *qooxdoo 1.5: Mobile Framework*. Juli 2011. – URL: http://manual.qooxdoo.org/1.5/pages/mobile/mobile_overview.html. (Zugriffsdatum: 14.11.2011)
- [Robbins 2010] Robbins, Jennifer N.; Lang, Jørgen W. (Hrsg.): *HTML & XHTML Taschen: kurz & gut (Taschenbibliothek (O'Reilly))*. 4. Auflage. O'Reilly Media, 2010. – ISBN 978-3-89721-558-0
- [Robinson 2010] Robinson, Tom: *iPhone Light Table*. 2010. – URL: <http://tlrobinson.net/projects/iphone-light-table/>. (Zugriffsdatum: 25.11.2011)
- [Rubine 1991] Rubine, Dean: Specifying gestures by example. In: *SIGGRAPH Comput. Graph.* 25 (1991), July, S. 329–337. – URL: <http://doi.acm.org/10.1145/127719.122753>. – ISSN 0097-8930
- [Rusadi 2009] Rusadi, Fajran I.: *Multitouch browser*. Juli 2009. – URL: <http://log.fajran.web.id/2009/06/05/multitouch-browser/>. (Zugriffsdatum: 20.01.2012). – TUIOjs und npTuiOClient
- [Saffer 2008] Saffer, Dan: *Designing Gestural Interfaces*. 1. O'Reilly Media, 11 2008. – ISBN 9780596518394
- [Samsung 2012] Samsung: *Samsung mobile - Smartphones*. 2012. – URL: <http://smartphones.samsung.de/>. (Zugriffsdatum: 25.01.2012)
- [Schäfer 2011a] Schäfer, Mathias: *Einführung in JavaScript*. 2011. – URL: <http://molily.de/js/>. (Zugriffsdatum: 18.08.2011)
- [Schäfer 2011b] Schäfer, Mathias: *"Für mehr Qualität im Web" - Webkrauts!: Was sind Web-Standards?* Aug. 2011. – URL: <http://www.webkrauts.de/2011/08/29/was-sind-webstandards/>. (Zugriffsdatum: 02.11.2011)

- [Schäfer und Strübig 2007] Schäfer, Mathias; Strübig, Joachim: *SELFHTML - Einführung in JavaScript und DOM*. 2007. – URL: <http://de.selfhtml.org/javascript/intro.htm>. (Zugriffsdatum: 14.11.2011)
- [Schmiedl et al. 2009] Schmiedl, Grischa; Seidl, Markus; Temper, Klaus: Mobile phone web browsing: a study on usage and usability of the mobile web. In: *Proceedings of the 11th International Conference on Human-Computer Interaction with Mobile Devices and Services*. New York, NY, USA : ACM, 2009 (MobileHCI '09), S. 70:1–70:2. – URL: <http://doi.acm.org/10.1145/1613858.1613942>. – ISBN 978-1-60558-281-8
- [Scholliers et al. 2011] Scholliers, Christophe; Hoste, Lode; Signer, Beat; De Meuter, Wolfgang: Midas: a declarative multi-touch interaction framework. In: *Proceedings of the fifth international conference on Tangible, embedded, and embodied interaction*. New York, NY, USA : ACM, 2011 (TEI '11), S. 49–56. – URL: <http://doi.acm.org/10.1145/1935701.1935712>. – ISBN 978-1-4503-0478-8
- [Schwab 2011] Schwab, Gregor: *Touchable-jQuery-Plugin*. Feb. 2011. – URL: <https://github.com/dotmaster/Touchable-jQuery-Plugin>. (Zugriffsdatum: 30.09.2011)
- [Seaward 2010] Seaward, Scott: *Jester - Simple Gesture Recognition in JavaScript*. Sept. 2010. – URL: <https://github.com/plainview/Jester>. (Zugriffsdatum: 01.11.2011)
- [Sencha 2011] Sencha: *Sencha Touch - Mobile JavaScript Framework*. 2011. – URL: <http://www.sencha.com/products/touch/>. (Zugriffsdatum: 30.09.2011)
- [Sibt-e-Hassan 2010] Sibt-e-Hassan, Zohaib: *moousture*. Jan. 2010. – URL: <http://maxpert.github.com/moousture>. (Zugriffsdatum: 30.09.2011)
- [Sizzle 2012] Sizzle: *Sizzle - JavaScript Selector Library*. 2012. – URL: <http://sizzlejs.com/>. (Zugriffsdatum: 10.01.2012)
- [Smus 2011a] Smus, Boris: *Developing for Multi-Touch Web Browsers*. April 2011. – URL: <http://www.html5rocks.com/en/mobile/touch.html>
- [Smus 2011b] Smus, Boris: *MagicTouch - Spec-compatible touch events based on npTuioClient callbacks*. April 2011. – URL: <https://github.com/borismus/MagicTouch>. (Zugriffsdatum: 03.11.2011)
- [Stefanov 2011] Stefanov, Stoyan: *JavaScript Patterns*. 1. O'Reilly Media, 2011. – ISBN 978-3-89721-598-6
- [Swigart 2005] Swigart, Scott: *Easily Write Custom Gesture Recognizers for Your Tablet PC Applications*. Nov. 2005. – URL: <http://msdn.microsoft.com/en-us/library/aa480673.aspx#tbconcustomgesturerecooverview>. (Zugriffsdatum: 12.08.2011)
- [Teiche et al. 2009] Teiche, Alex; Rai, Ashish K.; Yanc, Chris; Moore, Christian; Solms, Donovan; Çetin, Gökem; Riggio, Justin; Ramseyer, Nolan; D'Intino, Paul; Muller, Laurence; Khoshabeh, Ramsin; Bedi, Rishi; Bintahir, Mohammad T.; Hansen, Thomas; Roth, Tim; Sandler, Seth; Çetin, Gökem (Hrsg.); Bedi, Rishi (Hrsg.); Sandler, Seth (Hrsg.): *Multi-touch Technologies*. Natural User Interface Group, 5 2009
- [W3C 1999] W3C: *HTML 4.01 Specification: W3C Recommendation 24 December 1999*. Dec. 1999. – URL: <http://www.w3.org/TR/html401/>. (Zugriffsdatum: 17.10.2011)
- [W3C 2000] W3C; Pixley, Tom (Hrsg.); Corp., Netscape C. (Hrsg.): *Document Object Model (DOM) Level 2 Events Specification*. Nov. 2000. – URL: <http://www.w3.org/TR/DOM-Level-2-Events/events.html>. (Zugriffsdatum: 27.09.2011)

- [W3C 2010a] W3C: *Media Queries - W3C Candidate Recommendation 27 July 2010*. Juli 2010. – URL: <http://www.w3.org/TR/2010/CR-css3-mediaqueries-20100727/>. (Zugriffsdatum: 18.10.2011)
- [W3C 2010b] W3C; Schepers, Doug (Hrsg.): *Web Events Working Group Charter*. 10 2010. – URL: <http://www.w3.org/2010/webevents/charter/>. (Zugriffsdatum: 27.09.2011)
- [W3C 2011a] W3C: *CSS Basic User Interface Module Level 3 - Editor's Draft 29 October 2011*. Okt. 2011. – URL: <http://dev.w3.org/csswg/css3-ui/#pointer-events>. (Zugriffsdatum: 10.11.2011)
- [W3C 2011b] W3C: *DeviceOrientation Event Specification - W3C Working Draft 28 June 2011*. June 2011. – URL: <http://www.w3.org/TR/orientation-event/>. (Zugriffsdatum: 18.10.2011)
- [W3C 2011c] W3C; Schepers, Doug (Hrsg.); Rossi, Jacob (Hrsg.); Höhrmann, Microsoft B. (Hrsg.); Hégaret, Philippe L. (Hrsg.); Pixley, Tom (Hrsg.): *Document Object Model (DOM) Level 3 Events Specification - W3C Working Draft 31 May 2011*. 05. 2011. – URL: <http://www.w3.org/TR/2011/WD-DOM-Level-3-Events-20110531/>. (Zugriffsdatum: 30.09.2011)
- [W3C 2011d] W3C: *HTML Media Capture - W3C Working Draft 14 April 2011*. April 2011. – URL: <http://www.w3.org/TR/html-media-capture/>. (Zugriffsdatum: 08.02.2012)
- [W3C 2011e] W3C: *HTML5 HTML 5: A vocabulary and associated APIs for HTML and XHTML - W3C Working Draft 25 May 2011*. März 2011. – URL: <http://www.w3.org/TR/2011/WD-html5-20110525/>. (Zugriffsdatum: 17.10.2011)
- [W3C 2011f] W3C; Schepers, Doug (Hrsg.); Moon, Sangwhan (Hrsg.); Brubeck, Matt (Hrsg.): *Touch Events Specification*. 2011. – URL: <http://dvcs.w3.org/hg/webevents/raw-file/default/touchevents.html>. (Zugriffsdatum: 13.11.2011)
- [W3C 2011g] W3C; Schepers, Doug (Hrsg.); Moon, Sangwhan (Hrsg.); Brubeck, Matt (Hrsg.): *Touch Events version 1 - W3C Working Draft 13 September 2011*. Sept. 2011. – URL: <http://www.w3.org/TR/2011/WD-touch-events-20110913/>. (Zugriffsdatum: 27.09.2011)
- [W3C 2011h] W3C; Hickson, Ian (Hrsg.): *Web Workers - W3C Working Draft 01 September 2011*. Sept. 2011. – URL: <http://www.w3.org/TR/2011/WD-workers-20110901/>. (Zugriffsdatum: 10.02.2012)
- [Webel et al. 2008] Webel, Sabine; Keil, Jens; Zoellner, Michael: Multi-touch gestural interaction in X3D using hidden Markov models. In: *Proceedings of the 2008 ACM symposium on Virtual reality software and technology*. New York, NY, USA : ACM, 2008 (VRST '08), S. 263–264. – URL: <http://doi.acm.org/10.1145/1450579.1450644>. – ISBN 978-1-59593-951-7
- [Weil 2010] Weil, Florian: *Gestures for pointing devices in screen-based environments*. <http://www.derhess.de/>. Dez. 2010. – URL: http://www.derhess.de/sonstiges/paper_gesturedesign_print.pdf. (Zugriffsdatum: 11.02.2012)
- [Wigdor und Wixon 2011] Wigdor, Daniel; Wixon, Dennis: *Brave NUI World: Designing Natural User Interfaces for Touch and Gesture*. 1. Morgan Kaufmann, 4 2011. – ISBN 9780123822314
- [Wikipedia 2011] Wikipedia: *Acid (Browsertests)*. Nov. 2011. – URL: http://de.wikipedia.org/wiki/Acid_Browsertests. (Zugriffsdatum: 02.11.2011)
- [Wikipedia 2012] Wikipedia: *Webbrowser*. Feb. 2012. – URL: <http://de.wikipedia.org/wiki/Webbrowser>. (Zugriffsdatum: 11.02.2012)

- [Wobbrock et al. 2009] Wobbrock, Jacob O.; Morris, Meredith R.; Wilson, Andrew D.: User-defined gestures for surface computing. In: *Proceedings of the 27th international conference on Human factors in computing systems*. New York, NY, USA : ACM, 2009 (CHI '09), S. 1083–1092. – URL: <http://doi.acm.org/10.1145/1518701.1518866>. – ISBN 978-1-60558-246-7
- [Wobbrock et al. 2007] Wobbrock, Jacob O.; Wilson, Andrew D.; Li, Yang: Gestures without libraries, toolkits or training: a \$1 recognizer for user interface prototypes. In: *Proceedings of the 20th annual ACM symposium on User interface software and technology*. New York, NY, USA : ACM, 2007 (UIST '07), S. 159–168. – URL: <http://doi.acm.org/10.1145/1294211.1294238>. – ISBN 978-1-59593-679-0
- [Wood 2008] Wood, Daniel: *Methods for Multi-touch Gesture Recognition*. 2008. – URL: <http://people.cs.uct.ac.za/~nschiff/ipadGame/pdfs/daniel.pdf>. (Zugriffsdatum: 11.02.2012)
- [Wu et al. 2006] Wu, Mike; Shen, Chia; Ryall, Kathy; Forlines, Clifton; Balakrishnan, Ravin: Gesture Registration, Relaxation, and Reuse for Multi-Point Direct-Touch Surfaces. In: *Proceedings of the First IEEE International Workshop on Horizontal Interactive Human-Computer Systems*. Washington, DC, USA : IEEE Computer Society, 2006, S. 185–192. – URL: <http://dl.acm.org/citation.cfm?id=1109723.1110635>. – ISBN 0-7695-2494-X

ABBILDUNGSVERZEICHNIS

2.1	Beispiele von Eingabegeräten für die Interaktion mit Webinhalten	8
2.2	Einheiten des Ereignisprofils einer zweidimensionalen Geste	9
2.3	Design-Disziplinen nach [Cooper et al., 2007, S. xxxi] und zugeordnete Web-Standards	12
2.4	Box-Modell eines Web-Elements	14
2.5	Repräsentativer DOM-Baum eines HTML-Dokuments	14
2.6	Verbildlichung des Vererbungsprinzips in JavaScript anhand der Glasplattenmetapher nach [Debreuil, 2001]	16
2.7	Phasen der Event-Ausbreitung im DOM-Baum	19
3.1	Architekturmodelle nach [Echtler und Klinker, 2008], [Kammer et al., 2010] und [Khandkar und Maurer, 2010a] gegenüber einem webbasierten Ansatz eines Gestenerkennungs-Frameworks	21
3.2	MouseEvent- und TouchEvent-Interface nach [W3C, 2000] und [W3C, 2011g]	23
3.3	Implementierung eines webbasierten TUIO-Clients nach [Teiche et al., 2009], [Rusadi, 2009] und [Cailliau et al., 2009]	25
3.4	Showcases für die Verwendung von JavaScript-Frameworks für mobile Webanwendungen	27
3.5	Abbildung der Himmelsrichtungen nach [Sibt-e-Hassan, 2010]	29
3.6	Formale Beschreibung einer Kreisbewegung	29
3.7	Komponenten in TouchToolkit [Khandkar und Maurer, 2010b]	32
3.8	LASSO-Geste zur Auswahl einer Objektmenge	32
3.9	Midas Framework-Architektur [Scholliers et al., 2011]	33
3.10	Beispiele der Formalisierung mit GeForMT	34
3.11	Formalisierung des Objekt- und Aktionsbezugs [Henzen, 2010]	35
3.12	Modell eines künstlichen Neurons	37
3.13	Nicht-lineares Matching von Referenz- und Testmuster in DTW	37
3.14	Gegenüberstellung der Abstraktionsgrade nach [Kammer et al., 2010], [Cooper et al., 2007] und [W3C, 2010b]	45
3.15	Stiftbasierte Mehrfach-Selektion nach [Wigdor und Wixon, 2011]	46
3.16	SWIPE-Geste zum Blättern in Ansichten	47
3.17	Manipulation eines Video-Objektes [Cailliau et al., 2009]	47
3.18	Drehbares Kreisdiagramm mit Dojo Mobile [Dojo, 2011]	47
3.19	Touch Stack Widget nach [Murray, 2010]	47
3.20	Duplizieren durch Abreißen und Abziehen	48
3.21	Kontextmenü in Safari unter iOS	48
3.22	Löschen durch Durchkreuzen oder Ausradieren	48
3.23	Schleife nach links zum Widerrufen von Aktionen nach [Dorau, 2011, S. 160]	48
3.24	Bestätigen in einem Dialog	49
3.25	Gesture Search [Li, 2010a]	49
3.26	Ausschnitt aus dem Symbolalphabet in Graffiti [Blickenstorfer, 1995]	49
4.1	Übliche Bedienung eines Multitouch-Tablets	51
4.2	Bequemlichkeitszonen	52
4.3	Elemente des Ereignisprofils von Gesten	53

4.4	Beispiele für die Formalisierung der räumlichen Anordnung zusammengesetzter Gesten .	56
4.5	Spiralförmige Bewegung	57
4.6	Geschlossenes Rechteck	57
4.7	Formalisierung von Interaktionsbereichen zur Spezifizierung des Objektkontext	58
4.8	Herstellung des Objektbezugs von Gesten außerhalb des Erkennungsbereichs	59
5.1	Module im Namensraum GeForMT und verwendete Bibliotheken	68
5.2	Adaption der Parserimplementierung	69
5.3	Adaption der Selector Engine Sizzle	71
5.4	Zentrales Datenmodell	72
5.5	TemplateBuilder-Modul	73
5.6	Erkennung pfadbasierter Gesten basierend auf GeForMT und dem Verfahren nach [Wob- brock et al., 2007]	74
5.7	GestureRecognition-Modul	75
5.8	Zwei Zeichenflächen für die Visualisierung der Gestenpfade und aktuellen Kontaktpunkte über den Inhalten des Webdokumentes	77
5.9	Haupt-Programmierschnittstelle	78
5.10	Web-Interface der Testumgebung	80
5.11	Skizzierung der vordefinierten, sequentiell komponierten Gesten	81
5.12	Skizzierung der definierten Menge synchron ausgeführter Gesten	81
5.13	Skizzierung der objektorientierter Gesten zum Verschieben, Skalieren und Selektieren . .	81
5.14	Interface des Anwendungsprototypen	82

TABELLENVERZEICHNIS

2.1	Wichtige CSS-Selektoren im Überblick	13
3.1	Methoden der Gestenerkennung im Überblick	40
4.1	Zusammenfassung der Ansätze zur Beschreibung der räumlichen Anordnung zusammengesetzter Gesten	56
4.2	Übersicht der Optimierungs- und Erweiterungsvorschläge	64

LISTINGS

2.1	Allgemeiner Aufbau einer CSS-Regel	13
2.2	Beispiel eines Media-Queries für die optimierte Darstellung auf einem mobilen Endgerät	14
2.3	Objekt-Literal mit Getter- und Setter-Methoden	15
2.4	Konstruktoren und Prototypen	16
2.5	Revealing Module Pattern	17
2.6	Web Worker	18
2.7	API-Methoden für die Event-Registrierung nach DOM Level 2 [W3C, 2000]	18
3.1	Initialisierung von Multitouch-Gesten mit WKTouch [Gibson, 2011]	28
3.2	Einbindung von Multitouch-Gesten in Jester [Seaward, 2010]	29
3.3	Formalisierung und Registrierung einer Kreis-Geste in Mooousture [Sibt-e-Hassan, 2010]	29
3.4	LASSO-Geste in GDL [Khandkar und Maurer, 2010a]	32
3.5	Aufbau einer Regel am Beispiel der FLICK-Geste [Scholliers et al., 2011]	33
3.6	GeForMT-Syntax-Spezifikation in EBNF [Kammer et al., 2011]	35
4.1	Einschränkung der GeForMT-Syntax auf anwendbare Regeln für die Beschreibung von Maus- und Single-Touch-Gesten	54
4.2	Angepasste EBNF-Regel für Komposition von Gesten	55
4.3	Angepasste EBNF-Regeln für die Nutzung von Selektoren als Objektidentifikation	57
4.4	Angepasste EBNF-Regel für die Aktionszuweisung angelehnt an [Henzen, 2010, S. 83]	60
4.5	Aktionszuweisung basierend auf einem Objekt-Literal in JavaScript	61
4.6	Klammerung von wiederverwendeten Bezeichnern	62
4.7	Verwendung von Beschreibungsvorlagen für eine kontextunabhängige Formalisierung	63
5.1	Auszug der Grammatik nach PEG.js-Spezifikation für die Parser-Generierung	70
5.2	Einbindung des Frameworks in HTML	78
5.3	Initialisierung beim Laden des HTML-Dokumentes	79
5.4	Einbindung des Frameworks in HTML	79
5.5	Definition einer Geste und Registrierung	79